PENGELOMPOKAN DOKUMEN PETISI *ONLINE* DI SITUS CHANGE.ORG MENGGUNAKAN ALGORITME *HIERARCHICAL CLUSTERING* UPGMA

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:

Irwin Deriyan Ferdiansyah

145150201111007

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNIK INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

2018

PENGESAHAN

PENGELOMPOKAN DOKUMEN PETISI *ONLINE* DI SITUS CHANGE.ORG
MENGGUNAKAN ALGORITME *HIERARCHICAL CLUSTERING* UPGMA

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:
Irwin Deriyan Ferdiansyah
NIM: 145150201111007

Skripsi ini telah diuji dan dinyatakan lulus pada
15 Januari 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Sigit Adinugroho, S.Kom., M.Cs	M. Ali Fauzi, S.Kom, M.Kom
NIK: 201607 880701 1 000	NIK: 201502 890101 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D

NIP: 19710518 200312 1 001

IDENTITAS TIM PENGUJI

Penguji 1:

Nama : Fitra Abdurrachman Bachtiar, Dr.Eng., S.T, M.Eng

NIK : 2012018406281001

Email : fitra.bachtiar@ub.ac.id

Penguji 2:

Nama : Yuita Arum Sari, S.Kom., M.Kom

NIK : 2016098807152001

Email : yuita@ub.ac.id

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata di dalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 14 Januari 2018

Irwin Deriyan Ferdiansyah

NIM: 145150201111007

DAFTAR RIWAYAT HIDUP

Nama	: Irwin Deriyan Ferdiansyah
Tempat, Tgl Lahir	: Jember, 14 April 1996
Jenis Kelamin	: Laki-Laki
Agama	: Islam
Kewarganegaraan	: Indonesia
Status	: Belum Kawin
Hobi	: Olahraga Badminton dan Catur
Motto	: *Problem solving follow Priority*
Alamat Sekarang	: Jl. A. Yani no 30a Jenggawah - Jember
Telp	: 085336648080
Email	: Irwinderiyan@ymail.com

PENDIDIKAN :

- 2004 - 2008 **SD NEGERI 2 JENGGAWAH**
- 2008 - 2011 **SMP NEGERI 1 JEMBER**
- 2011 - 2014 **SMA NEGERI 4 JEMBER**
- 2014 - 2018 **UNIVERSITAS BRAWIJAYA**

PENGALAMAN PELATIHAN DAN ORGANISASI :

- 2016 **FAKULTAS ILMU KOMPUTER** SEBAGAI ASISTEN PRAKTIKUM PEMROGRAMAN LANJUT TAHUN 2016
- 2015 **UKM BRAWIJAYA CHESS CLUB** SEBAGAI STAFF DIVISI INVENTARIS
- 2016 **UKM BRAWIJAYA CHESS CLUB** SEBAGAI KEPALA DIVISI PERSONALIA
- 2015 **PANITIA TURNAMEN CATUR RAJA BRAWIJAYA III NASIONAL** SEBAGAI CO. DIVISI TRANSKOPER
- 2016 **PANITIA TURNAMEN CATUR RAJA BRAWIJAYA IV NASIONAL** SEBAGAI CO. DIVISI TRANSLOPER
- 2017 **UKM BRAWIJAYA CHESS CLUB** SEBAGAI KETUA UMUM

UCAPAN TERIMA KASIH

1. Almarhum bapak Winarno, S.Pd., Ibunda Tri Erni Rahayuningsih, S.Pd., kakak penulis Irwina Angelia Silvanasari, S.Kep., dan segenap keluarga besar yang selalu memberikan doa serta dukungan kepada penulis.
2. Bapak Sigit Adinugroho, S.Kom., M.Cs dan Bapak M. Ali Fauzi, S.Kom., M.Kom selaku dosen pembimbing satu dan dosen pembimbing dua selama pelaksanaan skripsi.
3. Bapak Wayan Firdaus, S.Kom., M.T., Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya Malang.
4. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D. selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya Malang.
5. Bapak M. Tanzil Furqon, S.Kom., M.Comp.Sc. selaku Sekretaris Jurusan Teknik Informatika.
6. Bapak Agus Wahyu Widodo, S.T., M.Cs. selaku Ketua Program Studi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya Malang.
7. Seluruh Dosen Teknik Informatika Universitas Brawijaya yang telah memberikan pelajaran dan ilmunya kepada penulis.
8. Seluruh Civitas Akademik Teknik Informatika Universitas Brawijaya yang telah membantu dan mendukung selama penulis menyelesaikan skripsi ini dan menempuh studi di Teknik Informatika Universitas Brawijaya.
9. Ambar Sukma Sekarina yang selalu menjadi motivasi dan penyemangat saat hadirnya kejenuhan dalam pengerjaan skripsi ini.
10. Seluruh keluarga besar UKM Catur Brawijaya Chess Club yang telah memberikan semangat dan juga motivasi dalam perjalanan memperoleh gelar sarjana baik secara langsung maupun secara tidak langsung.
11. Teman-teman grub line Curhat Power yang merupakan sahabat angkatan seperjuangan penulis.
12. Nella Kharisma dan Via Vallen untuk lagu-lagu penyemangat dan pelipur lara di setiap hari-hari pengerjaan skripsi.
13. Seluruh pihak yang telah membantu proses penulisan skripsi yang tidak dapat disebutkan satu per satu.

ABSTRAK

Change.org merupakan salah satu *website* yang sering digunakan oleh masyarakat untuk sarana penyampaian petisi dan kampanye sosial secara *online*. Kampanye lewat media sosial terbukti dapat menghasilkan perubahan. Aliran informasi petisi *online* yang berupa dokumen diperbarui setiap harinya dalam jumlah yang besar, membuat *cluster*ing dokumen menjadi sangat penting. *Cluster*ing dokumen adalah proses pengelompokan dokumen yang memiliki kesamaan topik. Tujuannya untuk membagi dokumen berdasarkan kesamaan, sehingga memudahkan dalam proses pencarian. Metode yang digunakan adalah *hierarchical cluster*ing UPGMA atau *Unweighted Pair-Group Method using Arithmetic averages* dengan menambahkan reduksi fitur menggunakan metode *Latent Semantic Indexing* hasil pemecahan matrik *Singular Value Decomposition*. Hasil penelitian menyimpulkan bahwa *Latent Semantic Indexing* dapat mengatasi permasalahan pada data berdimensi tinggi. Data yang digunakan berjumlah 100 petisi. Dari hasil pengujian performansi menggunakan *Cophenetic Correlation Coefficient* diperoleh nilai *cophenetic* sebesar 0,75959 pada *rank* matrik LSI sebanyak 10% dan *Silhouette Coefficient* sebesar 0,36862 dengan jumlah *cluster* sebanyak 2 *cluster*.

Kata kunci: pengelompokan dokumen, Change.org, UPGMA *clustering*, *Singular Value Decomposition*, *Latent Semantic Indexing, Silhouette Coefficient, Cophenetic Correlation Coefficient*.

ABSTRACT

Change.org is a website that is often used by people, which means for online delivering petitions and social campaignings. Campaign through social media had been proven that can make a change. The flow information of online petitions documents is updated daily in large numbers. It makes documents clustering being very important. Documents clustering is a process of grouping documents which have same topic. It aims to devide documents by its similarly, so the process of searching will be easier. This study uses hierarchical clustering UPGMA or unweighted pair-group method by arithmetic averages with adding feature reduction using Latent Semantic Indexing method, that is the result of splitting Singular Value Decomposition matrix. The result of this study conclude that Latent Semantic Indexing method can solved the problem in high-dimensional data. The data conducted by 100 petitions. The result of performance testing which used Cophenetic Correlation Coefficient obtained conphenetic value of 0.75959 at LSI matrix rank of 10 % and Silhouette Coefficient of 0.36862 with number of clusters as many as 2 clusters.

***Keywords**: Document Clustering, Change.org, UPGMA clustering, Singular Value Decomposition, Latent Semantic Indexing, Silhouette Coefficient, Cophenetic Correlation Coefficient*.

.

PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa atas limpahan rahmat serta hidayahnya yang telah diberikan, sehingga skripsi dengan judul " Pengelompokan Dokumen Petisi *Online* di Situs Change.Org Menggunakan Algoritme *Hierarchical Clustering* UPGMA" telah diselesaikan dengan baik. Skripsi ini tentunya tidak dapat diselesaikan tanpa adanya bantuan dari berbagai pihak. Semoga Tuhan Yang Maha Esa memberikan balasan sesuai dengan jasa-jasa yang telah diberikan kepada penulis selama ini. Skripsi ini membahas tentang sistem yang mampu menerapkan algoritme UPGMA *cluster*ing untuk proses pengelompokan dokumen petisi *online* pada situs change.org yang cukup digemari oleh masyarakat saat ini. Penulis menyadari bahwa skripsi ini memiliki banyak kekurangan pada format penulisan maupun isi dari skripsi ini, maka dari itu penulis mengharapkan kritik dan saran yang bersifat membangun dari para pembaca. Semoga skripsi ini dapat memberikan informasi yang bermanfaat bagi semua pihak dan kalangan.

Malang, 14 Januari 2018

Penulis

Irwinderiyan@ymail.com

DAFTAR ISI

PENGESAHAN ii
PERNYATAAN ORISINALITAS iii
KATA PENGANTAR iv
ABSTRAK vi
ABSTRACT vii
DAFTAR ISI viii
DAFTAR TABEL xii
DAFTAR GAMBAR xiv
DAFTAR PERSAMAAN xvi
DAFTAR KODE PROGRAM xvii
DAFTAR LAMPIRAN xviii
BAB 1 PENDAHULUAN 1
1.1 Latar belakang 1
1.2 Rumusan masalah 3
1.3 Tujuan 3
1.4 Manfaat 3
1.5 Batasan masalah 4
1.6 Sistematika penulisan 4
BAB 2 TINJAUAN PUSTAKA 6
2.1 Tinjauan Penelitian 6
2.2 Petisi 8
2.3 Change.org 8
2.4 *Data Mining* 9
2.5 *Text Mining* 9
2.6 *Text Pre-processing* 9
2.6.1 *Case Folding* dan *Tokenizing* 9
2.6.2 *Filtering* 10
2.6.3 *Stemming* 11
2.7 *Term Weighting* 12
2.7.1 Pembobotan TFIDF 13
2.8 *Singular Value Decomposition* 14

2.9 *Latent Semantic Indexing* (LSI) 15
2.10 *Vector Space Model* (VSM) 16
2.11 *Clustering* 16
2.11.1 UPGMA (*Unweighted Pair Group Method with Arithmetic Mean)* 17
2.12 Evaluasi 17
2.12.1 *Cophenetic Correlation Coefficient* 18
2.12.2 *Silhouette Coefficient* 18
BAB 3 METODOLOGI 20
3.1 Studi Pustaka 21
3.2 Analisis Kebutuhan 21
3.3 Pengumpulan Data 21
3.4 Perancangan Sistem 21
3.5 Implementasi Sistem 22
3.6 Pengujian 23
3.7 Kesimpulan 23
BAB 4 PERANCANGAN 24
4.1 *Text Preprocessing* 25
4.1.1 *Case Folding* dan *Tokenizing* 26
4.1.2 *Filtering* 27
4.1.3 *Stemming* 28
4.2 Pembobotan TFIDF 30
4.2.1 Menghitung *term* frequency 30
4.2.2 Menghitung inverse document frequency 32
4.2.3 Menghitung TFIDF 33
4.3 *Singular Value Decomposition* 33
4.4 *Latent Semantic Indexing* 34
4.5 *Vector Space Model* (VSM) 37
4.6 *Cluster*ing UPGMA 38
4.7 Manualisasi Perhitungan Data 41
4.8 Perancangan Antarmuka Sistem 59
4.9 Perancangan Pengujian 60
4.9.1 Rancangan Pengujian *Cophenetic Correlation Coefficient* 60

4.9.2 Rancangan Pengujian *Silhouette Coefficient* 61
4.10 Penarikan Kesimpulan 62
BAB 5 IMPLEMENTASI 63
5.1 Spesifikasi sistem 63
5.1.1 Spesifikasi Perangkat Lunak 63
5.1.2 Spesifikasi Perangkat Keras 63
5.2 Implementasi Program 64
5.3 *Text Preprocessing* 67
5.3.1 Fungsi *lowercase* 67
5.3.2 Fungsi hapusnonabjad 68
5.3.3 Fungsi *Tokenizing* 68
5.3.4 Fungsi *Filtering* 69
5.3.5 Fungsi *Stemming*nazief 69
5.4 Pembobotan TFIDF 70
5.4.1 Fungsi pembentukan*term*unik 71
5.4.2 Fungsi rawTF 71
5.4.3 Fungsi pembobotanTF 72
5.4.4 Fungsi perhitunganDF 73
5.4.5 Fungsi pembobotanIDF 74
5.4.6 Fungsi pembobotanTFIDF 75
5.5 *Singular Value Decomposition* 75
5.5.1 Fungsi SVD 76
5.6 *Latent Semantic Indexing* 77
5.6.1 Fungsi reducematrixLSI 77
5.7 *Cosine Similarity* 79
5.7.1 Fungsi hitungcosinesimilarity 79
5.8 *Clustering* UPGMA 80
5.8.1 Fungsi caridekat 80
5.8.2 Fungsi update 81
5.8.3 Fungsi updatearray 82
5.8.4 Fungsi jarak 82
5.8.5 Fungsi ratarata 83

5.9 *Cophenetic Correlation Coefficient* 84
5.9.1 Fungsi *copheneticcorrelation* 84
5.10 *Silhouette Coefficient* 85
5.10.1 Fungsi evaluasi*Silhouette* 85
5.10.2 Fungsi hitungai 87
5.10.3 Fungsi hitungbi 88
5.11 Implementasi Antarmuka 88
BAB 6 PENGUJIAN DAN ANALISIS 104
6.1 Pengujian *Cophenetic Correlation Coefficient* 104
6.1.1 Skenario pengujian *Cophenetic Correlation Coefficient* 104
6.1.2 Analisis pengujian *Cophenetic Correlation Coefficient* 104
6.2 Pengujian *Silhouette Coefficient* 109
6.2.1 Skenario pengujian *Silhouette Coefficient* 109
6.2.2 Analisis pengujian *Silhouette Coefficient* 112
BAB 7 Penutup 115
7.1 Kesimpulan 115
7.2 Saran 115
DAFTAR PUSTAKA 117
DAFTAR LAMPIRAN 119

DAFTAR TABEL

Tabel 2.1 Ringkasan *dataset* untuk evaluasi beberapa kriteria *clustering* 6
Tabel 2.2 Hasil Evaluasi *Agglomerative clustering* dengan *Fscore* 7
Tabel 2.3 Hasil Evaluasi *Partitional clustering* dengan *Fscore* 7
Tabel 2.4 Hasil Evaluasi *Agglomerative clustering* dengan *entropy* 7
Tabel 2.5 Hasil Evaluasi *Partitional clustering* dengan *entropy* 7
Tabel 4.1 Data Petisi *Online* 41
Tabel 4.2 Manualisasi proses *case folding* 41
Tabel 4.3 Manualisasi proses *Tokenizing* 42
Tabel 4.4 Manualisasi proses *Filtering* 43
Tabel 4.5 Manualisasi proses *Stemming* 44
Tabel 4.6 Manualisasi perhitungan frekuensi dokumen 44
Tabel 4.7 Manualisasi perhitungan $W_{TF\ (t,d)}$ dan $W_{IDF\ (t,d)}$ 47
Tabel 4.8 Manualisasi perhitungan pembobotan TFIDF 49
Tabel 4.9 Hasil perhitungan *Singular Value Decomposition* matrik *U* 52
Tabel 4.10 Hasil perhitungan *Singular Value Decomposition* matrik *S* 55
Tabel 4.11 Hasil perhitungan *Singular Value Decomposition* matrik V^T 55
Tabel 4.12 Hasil pemotongan matrik pada matrik *S* 55
Tabel 4.13 Hasil pemotongan matrik pada matrik V^T 56
Tabel 4.14 Hasil perhitungan reduksi matrik 56
Tabel 4.15 Hasil perhitungan *cosine similarity* 57
Tabel 4.16 Hasil pembentukan *cluster* iterasi 1 58
Tabel 4.17 Hasil pembentukan *cluster* iterasi 2 58
Tabel 4.18 Hasil pembentukan *cluster* iterasi 3 58
Tabel 4.19 Hasil pembentukan *cluster* iterasi 4 58
Tabel 4.20 Hasil pembentukan *cluster* iterasi 5 59
Tabel 4.21 Rancangan pengujian *Cophenetic Correlation Coefficient* 60
Tabel 4.22 Rancangan pengujian *Silhouette Coefficient* 61
Tabel 5.1 Spesifikasi Perangkat Lunak 63
Tabel 5.2 Spesifikasi Perangkat Keras 63
Tabel 5.3 Daftar Fungsi Implementasi Program 64
Tabel 6.1 Pengujian Nilai *Cophenetic Correlation Coefficient* 104

Tabel 6.2 Pengujian Nilai *Silhouette* pada *rank* matrik LSI 10% 109
Tabel 6.3 Nilai *Silhouette* setiap dokumen *rank* LSI 10% pada level 2 *cluster* 112

DAFTAR GAMBAR

Gambar 2.1 Case folding 10
Gambar 2.2 *Tokenizing* 10
Gambar 2.3 *Filtering* 11
Gambar 2.4 *Stemming* 11
Gambar 2.5 Pemotongan matrik pada SVD 15
Gambar 2.6 Pemotongan matik LSI 15
Gambar 3.1 Diagram Blok Metodologi Penelitian 20
Gambar 3.2 Diagram Perancangan Sistem 22
Gambar 4.1 Diagram alir sistem 24
Gambar 4.2 Diagram alir *text preprocesssing* 25
Gambar 4.3 Diagram alir *case folding* 26
Gambar 4.4 Diagram alir *Tokenizing* 27
Gambar 4.5 Diagram alir *Filtering* 28
Gambar 4.6 Diagram alir *Stemming* 29
Gambar 4.7 Diagram alir Pembobotan TFIDF 30
Gambar 4.8 Diagram alir perhitungan TF 31
Gambar 4.9 Diagram alir perhitungan IDF 32
Gambar 4.10 Diagram alir perhitungan TFIDF 33
Gambar 4.11 Diagram alir *Singular Value Decomposition* 34
Gambar 4.12 Diagram pemotongan matrik S 35
Gambar 4.13 Diagram pemotongan matrik V^T 36
Gambar 4.14 Diagram alir *Latent Semantic Indexing* 37
Gambar 4.15 Diagram alir *cosine similarity* 38
Gambar 4.16 Diagram alir *clustering* UPGMA 39
Gambar 4.17 Diagram alir perhitungan jarak terdekat antar dokumen 40
Gambar 4.18 Perancangan antarmuka sistem 59
Gambar 5.1 Implementasi Antarmuka *Input* Data 89
Gambar 5.2 Implementasi Antarmuka Proses Data 90
Gambar 5.3 Implementasi Antarmuka Lowecase 91
Gambar 5.4 Implementasi Antarmuka Hapus Karakter Non Abjad 91
Gambar 5.5 Implementasi Antarmuka *Tokenizing* 92

Gambar 5.6 Implementasi Antarmuka *Filtering* 92
Gambar 5.7 Implementasi Antarmuka *Stemming* 93
Gambar 5.8 Implementasi Antarmuka *Term* Frekuensi 94
Gambar 5.9 Implementasi Antarmuka Pembobotan *Term* Frekuensi 94
Gambar 5.10 Implementasi Antarmuka Perhitungan DF 95
Gambar 5.11 Implementasi Antarmuka Pembobotan IDF 95
Gambar 5.12 Implementasi Antarmuka Pembobotan TFIDF 96
Gambar 5.13 Implementasi Antarmuka Pembentukan Matrik U 96
Gambar 5.14 Implementasi Antarmuka Pembentukan Matrik S 96
Gambar 5.15 Implementasi Antarmuka Pembentukan Matrik V^T 97
Gambar 5.16 Implementasi Antarmuka Reduksi Matrik V^T 98
Gambar 5.17 Implementasi Antarmuka Reduksi Matrik S 99
Gambar 5.18 Implementasi Antarmuka Reduksi Matrik V 99
Gambar 5.19 Implementasi Antarmuka Matrik LSI 100
Gambar 5.20 Implementasi Antarmuka Cosine Similarity 101
Gambar 5.21 Implementasi Antarmuka *Clustering* UPGMA 102
Gambar 5.22 Implementasi Antarmuka Evaluasi *Silhouette Coefficient* 102
Gambar 5.23 Implementasi Antarmuka Evaluasi *Cophenetic* 103
Gambar 6.1 Grafik Pengujian *Cophenetic Correlation Coefficient* 106
Gambar 6.2 *Histogram* data dengan *cluster* dan tanpa *cluster* 108
Gambar 6.3 *Histogram* probabilitas kepadatan data 100 petisi *online* 108
Gambar 6.4 Grafik Pengujian *Silhouette Coefficient* 111
Gambar 6.5 Rata-rata nilai *Silhouette* pada level 2 *cluster* 114

BAB 1 PENDAHULUAN

1.1 Latar belakang

Perkembangan media elektronik khususnya teknologi informasi saat ini sangatlah cepat terutama internet. Internet sendiri oleh beberapa kalangan sudah dijadikan sebagai wadah tukar pikiran dan juga aspirasi untuk mencapai tujuan tertentu. Dari sektor pemerintah , internet juga sudah dijadikan sebagai penyalur aspirasi dan komunikasi dua arah dengan masyarakat. Di Indonesia salah satu media yang bisa digunakan masyarakat untuk menyuarakan aspirasi dan pengaduannya terhadap kinerja pemerintahan, ataupun pihak-pihak lain yang bersangkutan adalah melalui petisi *online*. Petisi *online* muncul sebagai alat yang ampuh bagi masyarakat untuk memberikan dampak positif serta kemudahan individu dan kelompok untuk mengadakan petisi secara *online*, dengan adanya petisi *online* maka masyarakat ditawarkan jangkauan akses yang lebih luas dalam periode waktu yang lebih singkat. (Mustikaningsih, 2016)

Change.org merupakan salah satu *website* yang kerap digunakan oleh masyarakat untuk sarana penyampaian petisi dan kampanye sosial secara *online*. Kampanye lewat media sosial terbukti dapat menghasilkan perubahan. Media sosial digunakan untuk tempat berkumpul virtual dan secara tak langsung dapat memobilisasi massa yang memiliki tujuan sama (Wahyudi ,2012). Situs ini telah ada di indonesia lebih dari 5 tahun. Sejak awal mula 2012 didirikan di indonesia , jumlah pengguna telah tumbuh dari 10 ribu sampai lebih dari 3,5 juta pengguna diseluruh dunia. Dan banyak memenangkan petisi -petisi yang berisikan tentang penyelamatkan hutan, pemulihan hak suara, pembelaan hak penyandang cacat bahkan sampai melindungi satwa liar (Change.org inc , 2017).

Antusiasme masyarakat dunia khususnya indonesia untuk membuat dan mendukung petisi – petisi yang ada pada situs change.org cukuplah besar. Besarnya antusiasme masyarakat sayangnya kurang didukung dengan fasilitas yang memadai di dalam situs ini. Pencarian atas petisi *online* pada situs ini hanya di dasari pada kata kunci yang dimasukkan saja. Belum tertata atau terkelompokannya petisi - petisi yang ada pada situs ini mengakibatkan tidak adanya fitur tambahan mengenai topik-topik tertentu yang dapat dipilih dan memudahkan pengguna untuk memilih petisi yang diinginkan. Aliran informasi petisi *online* yang berupa dokumen diperbarui setiap harinya dalam jumlah yang besar, membuat *clustering* dokumen menjadi sangat penting. *Clustering* dokumen adalah proses pengelompokan dokumen yang memiliki kesamaan topik. Tujuan dari proses *clustering* ini membagi dokumen berdasarkan kesamaan, sehingga memudahkan dalam proses pencarian. (Milatina, Syukur dan Supriyanto, 2012).

Clustering dokumen sudah sering digunakan untuk memudahkan pengguna dalam pencarian dokumen. Hal ini dikarenakan dokumen yang relevan akan cenderung berada pada kelompok yang sama jika koleksi dokumen dilakukan *clustering*. Beberapa penelitian sebelumnya yang berkaitan dengan *clustering* dokumen antara lain penelitian yang dilakukan oleh Husni, Dwi dan Syarief (2015)

di mana pada penelitian tersebut mengelompokkan dokumen web berupa berita berbahasa Indonesia dengan menggunakan Algoritme K-Means, dan menjelaskan bahwa dokumen web yang sudah terkelompokkan dengan baik, akan memberikan akurasi yang memuaskan dalam hal pencarian. Selanjutnya adalah penelitian yang dilakukan oleh Rahadian et al (2017) di mana menganalisis judul majalah Kawanku menggunakan algoritme *clustering* yang disimulasi dengan konsep *Big data*, menyimpulkan bahwa penggunaan data yang besar membutuhkan suatu pengolahan, pengelompokan dan pengkomunikasian antar data dokumen menggunakan *text mining*.

Pada *clustering* teks terdapat suatu permasalahan yaitu dengan adanya fitur – fitur yang berdimensi tinggi atau data berdimensi tinggi. Data berdimensi tinggi merupakan data yang memiliki fitur berjumlah banyak. Hal ini bisa disebabkan karena adanya data yang tidak relevan dan redundan. Proses dari *cluster*ing akan kurang optimal jika didalamnya masih terdapat fitur yang tidak relevan dan redundan. Oleh karena itu diperlukan metode yang dapat mengurangi dimensi pada fitur berdimensi tinggi. (Langgeni, Baizal Dan Firdaus, 2010). Terdapat cara untuk dapat membatasi jumlah fitur yang terlibat pada data berdimensi tinggi, yaitu dengan mereduksi matrik *term* dokumen menggunakan *Latent Semantic Indexing* (LSI). *Latent Semantic Indexing* adalah suatu metode untuk mendapatkan pola dalam suatu koleksi dokumen. Pada dasarnya, LSI menggunakan metode *Singular Value Decomposition* (SVD) untuk memperlihatkan hubungan yang mendasari *term* dan dokumen dalam semua kombinasi yang memungkinkan dan membuang *noise* yang ada pada ruang vektor. Pada hasil penelitian yang dilakukan Muflikhah dan Baharudin (2009) menyatakan bahwa reduksi dimensi dengan metode LSI dapat memperoleh performa lebih tinggi dari pada tanpa menggunakan metode LSI.

Berdasarkan kebutuhan akan pengelompokan petisi *online* pada situs ini, maka diperlukannya suatu analisis *clustering* yang sesuai dengan data petisi *online* di situs change.org. Ada dua klasifikasi metode *clustering* yang berkembang saat ini, yaitu *hierarchical cluster*ing dan *non hierarchical clustering* (*partitional clustering*). Perbedaan utama antara metode *hierarchical clustering* dan *non hierarchical clustering* terletak pada nilai k atau jumlah *cluster* yang akan dibentuk. Pada *hierarchical clustering*, tidak dilakukan penentuan banyaknya nilai *k* di awal proses, *clustering* dibentuk secara hirarki berbentuk *dendrogram.* Sebaliknya untuk *partitional clustering*, dibutuhkan jumlah nilai *k* di awal dan penentuan *centroid* atau pusat *cluster*.

Algoritme *hierarchical clustering* lebih sesuai diterapkan pada situs ini karena pada algoritme tersebut, dokumen yang akan dikelompokkan belum memiliki jumlah *cluster* sebelumnya. Keunggulan *hierarchical clustering* adalah kemudahan dalam menangani bentuk-bentuk kesamaan atau jarak antar *cluster*, lebih efisien dari segi komputasi tetapi tetap memberikan hasil yang bagus dan mudah dalam pengaplikasiannya daripada *partitional clustering*. Ada beberapa pilihan dalam *hierarchical clustering* yang bisa digunakan, penelitian ini menggunakan algoritme UPGMA karena menurut penelitian yang dilakukan oleh Zhao dan Karypis (2005),

UPGMA atau *Unweighted Pair-Group Method using Arithmetic averages* memiliki keunggulan dari dua algoritme *hierarchical clustering* umum lainnya, yaitu *single linkage* dan *complete linkage*. Pada penelitian tersebut, dilakukan analisis membandingkan algoritme *hierarchical clustering* berdasarkan nilai *Fscore* dan *Entropy*. Dan menghasilkan nilai rata-rata *Fscore* 0,929 dan nilai *Entropy* 1,277 pada UPGMA. Pada kedua evaluasi *clustering* tersebut, semakin tinggi Nilai *Fscore* menandakan algoritme tersebut lebih baik, sedangkan Nilai *Entropy* menandakan sebaliknya, semakin rendah nilai *Entropy* maka semakin baik algoritme tersebut. Nilai rata-rata *Fscore* dari *single linkage* dan *complete linkage* adalah 0,649 dan 0,760. Nilai rata-rata *Entropy single linkage* dan *complete linkage* adalah 3,589 dan 1,340. Maka dapat diambil kesimpulan dari penelitian tersebut bahwa algoritme *hierarchical clustering* UPGMA memiliki keunggulan dalam hal evaluasi berdasarkan nilai rata-rata entropy dan *Fscore*.

Berdasarkan penjelasan pada latar belakang tersebut, maka peneliti mengambil judul penelitian "pengelompokan dokumen petisi *online* di situs change.org menggunakan algoritme *hierarchical clustering* UPGMA".

1.2 Rumusan masalah

Mengacu pada penjelasan latar belakang di atas, rumusan masalah yang dapat dikaji berupa

1. Bagaimana menerapkan algoritme *hierarchical clustering* UPGMA untuk mengelompokkan petisi *online* di situs change.org?
2. Bagaimana pengaruh reduksi matrik menggunakan *Latent Semantic Indexing* pada hasil pengelompokan petisi *online* disitus change.org?
3. Bagaimana performa dari algoritme *hierarchical clustering* UPGMA untuk pengelompokan petisi *online* di situs change.org?

1.3 Tujuan

Penelitian yang dilakukan oleh penulis memiliki tujuan yaitu:

1. Menerapkan algoritme *hierarchical clustering* UPGMA untuk mengelompokkan petisi *online* pada situs change.org.
2. Mengetahui pengaruh reduksi matrik menggunakan *Latent Semantic Indexing* pada pengelompokan petisi *online* dengan algoritme *hierarchical clustering* UPGMA pada situs change.org
3. Mengetahui performa dari proses pengelompokan petisi *online* dengan algoritme *hierarchical clustering* UPGMA pada situs change.org.

1.4 Manfaat

Adapun manfaat dari penelitian ini adalah petisi *online* pada situs change.org yang merupakan data *text* tidak terstruktur dapat diorganisasikan dengan baik sehingga di harapkan waktu proses pengambilan informasi pada situs tersebut dapat lebih efisien dan lebih terstruktur. Dengan adanya penelitian ini di harapkan juga dapat mempercepat dan membantu dalam memberikan informasi mengenai

keragaman topik yang ada pada petisi *online* di situs change.org guna untuk mendukung dan ikut serta dalam menyuarakan aspirasi serta mengkritisi kinerja yang dianggap kurang dari pemerintah hingga permasalah-permasalahan lain yang dianggap sesuai untuk dipetisikan, khususnya di indonesia. Di samping itu, diharapkan dapat dimanfaatkan oleh pihak change.org Indonesia untuk penambahan fitur berdasarkan topik hasil pengelompokan pada penelitian ini, supaya memudahkan pengguna petisi *online* dalam pencarian petisi yang dibutuhkan.

1.5 Batasan masalah

Hal-hal yang menjadi batasan masalah pada penelitian ini yaitu:

1. Data petisi yang digunakan adalah data petisi *online* yang dituliskan pada situs Change.org.
2. Data petisi yang diambil merupakan data petisi berbahasa Indonesia.
3. Data petisi yang digunakan adalah data antara bulan januari 2013 hingga Juli 2017.
4. Data petisi yang digunakan adalah sejumlah 100 dokumen.
5. Sistem tidak memperhatikan kesalahan kata pada dokumen petisi *online.*
6. Implementasi aplikasi *clustering* petisi *online* menggunakan Bahasa pemrograman *JAVA.*

1.6 Sistematika penulisan

1. BAB I PENDAHULUAN

Latar belakang, rumusan masalah, tujuan penelitian, manfaat penelitian dan batasan masalah yang terkait dengan implementasi algoritme *hierarchical clustering* UPGMA untuk pengelompokan dokumen pada situs change.org.

2. BAB II KAJIAN PUSTAKA

Teori dasar pendukung implementasi antara lain mengenai *text* mining dan data mining yang dijadikan dasar ilmu *cluster*ing dengan algoritme hierarchical *cluster*ing UPGMA.

3. BAB III METODOLOGI

Metode dan langkah kerja yang dilakukan dalam penelitian implementasi algoritme *hierarchical cluster*ing UPGMA untuk pengelompokan dokumen pada situs change.org.

4. BAB IV PERANCANGAN

Perancangan sistem serta proses–proses algoritme *hierarchical cluster*ing UPGMA untuk pengelompokan dokumen dari situs change.org, contoh perhitungan manual dan perancangan antarmuka sistem.

5. BAB V IMPLEMENTASI

Menjelaskan tentang implementasi pada sistem. Implementasi sistem berisikan spesifikasi sistem, implementasi program, dan implementasi antarmuka.

6. BAB VI PENGUJIAN DAN ANALISIS

Bab ini berisikan pengujian dari algoritme *Hierarchical clustering* UPGMA beserta analisis hasil yang diperoleh.

7. BAB VII PENUTUP

Bab ini berisi tentang kesimpulan dan saran.

BAB 2 TINJAUAN PUSTAKA

Pada bab ini akan membahas beberapa teori dan kajian pustaka yang bersangkutan dengan pengelompokan dokumen petisi *online* menggunakan algoritme *hierarchical clustering* UPGMA (*Unweighted Pair Group Method with Arithmetic Mean*). Dasar teori akan membahas teori mengenai petisi, *data mining*, *text mining* dan beberapa hal yang berkaitan pada penelitian ini.

2.1 Tinjauan Penelitian

Tinjauan utama pada penelitian ini merujuk pada 3 penelitian utama, yang pertama merupakan penelitian yang telah dilakukan oleh Zhao dan Karypis (2005) yang berjudul *Hierarchical clustering Algorithms for Document Datasets*. Pada penelitian tersebut dilakukan penelitian pada *dataset* dokumen yang menggunakan sebelas *dataset* yang berbeda untuk mengevaluasi beberapa kriteria *clustering* yang meliputi *partitional clustering* dan *agglomerative clustering*. Tabel 2.1 merupakan ringkasan dari *dataset* penelitian tersebut.

Tabel 2.1 Ringkasan *dataset* untuk evaluasi beberapa kriteria *clustering*

Data	Sumber	Jumlah Dokumen	Jumlah *Term*	Jumlah Kelas
fbis	FBIS (TREC)	2463	12674	17
hitech	San Jose Mercury (TREC)	2301	13170	6
reviews	San Jose Mercury (TREC)	4069	23220	5
la1	LA Times (TREC)	3204	21604	6
tr31	TREC	927	10128	7
tr41	TREC	878	7454	10
re0	Reuters-21578	1504	2886	13
rel	Reuters-21578	1657	3758	25
k1a	WebACE	2340	13879	20
k1b	WebACE	2340	13879	6
wap	WebACE	1560	8460	20

Sumber : Zhao dan Karypis (2005)

Penelitian tersebut mengevaluasi 9 algoritme *agglomerative clustering* dengan 6 algoritme *partitional clustering*. Dari 9 Algoritme *agglomerative clustering* tersebut terdapat 3 algoritme yang dikenal memiliki akurasi tinggi yaitu *single linked, complete linked* dan UPGMA (*Unweighted Pair Group Method with Arithmetic Mean*). Pada pengevaluasian 15 macam algoritme *clustering* tersebut, dilakukan evaluasi dengan menggunakan nilai *Fscore* dan nilai *entropy*. Pada evaluasi nilai *Fscore*, algoritme *clustering* yang tertinggi menunjukan hasil yang terbaik, sedangkan untuk nilai *entropy*, algortima *clustering* yang terendah yang di nilai terbaik pada evaluasi tersebut. Hasilnya menunjukan pada evaluasi algoritme *agglomerative clustering*, UPGMA memperoleh hasil yang terbaik dari ke 8 algoritme *agglomerative* lainnya. UPGMA menghasilkan nilai rata-rata *Fscore*

0,929 dan nilai rata-rata *Entropy* 1,277. Sedangkan untuk algoritme *partitional clustering* menunjukan algoritme $p\mathcal{I}_2$ terbaik dibandingkan 5 algoritme *partitional clustering* lainnya dengan nilai rata-rata *Fscore* 0,987 dan nilai rata-rata *entropy* 1,027. Tabel 2.2 sampai 2.5 merupakan keseluruhan hasil evaluasi *clustering* dengan rata-rata *Fscore* dan rata-rata *entropy*.

Tabel 2.2 Hasil Evaluasi *Agglomerative clustering* dengan *Fscore*

Agglomerative Clustering									
	$\mathcal{E}_1$	$\mathcal{G}_1$	$\mathcal{H}_1$	$\mathcal{H}_2$	$\mathcal{I}_1$	$\mathcal{I}_2$	**UPGMA**	**slink**	**clink**
Rata-rata	0,855	0,855	0,889	0,879	0,836	0,89	**0,929**	0,649	0,76

Sumber: Zhao dan Karypis (2005)

Tabel 2.3 Hasil Evaluasi *Partitional clustering* dengan *Fscore*

Partitional Clustering						
	$p\mathcal{E}_1$	$p\mathcal{G}_1$	$p\mathcal{H}_1$	$p\mathcal{H}_2$	$p\mathcal{I}_1$	$p\mathcal{I}_2$
Rata-rata	0,968	0,971	0,972	0,978	0,932	**0,987**

Sumber: Zhao dan Karypis (2005)

Tabel 2.4 Hasil Evaluasi *Agglomerative clustering* dengan *entropy*

Agglomerative Clustering									
	$\mathcal{E}_1$	$\mathcal{G}_1$	$\mathcal{H}_1$	$\mathcal{H}_2$	$\mathcal{I}_1$	$\mathcal{I}_2$	**UPGMA**	**slink**	**clink**
Rata-rata	1,381	1,469	1,381	1,370	1,434	1,369	**1,277**	3,589	1,340

Sumber: Zhao dan Karypis (2005)

Tabel 2.5 Hasil Evaluasi *Partitional clustering* dengan *entropy*

Partitional Clustering						
	$p\mathcal{E}_1$	$p\mathcal{G}_1$	$p\mathcal{H}_1$	$p\mathcal{H}_2$	$p\mathcal{I}_1$	$p\mathcal{I}_2$
Rata-rata	1,046	1,035	1,072	1,039	1,186	**1,027**

Sumber: Zhao dan Karypis (2005)

Berdasarkan tinjauan tersebut penelitian ini menerapkan algoritme UPGMA dalam pengelompokan dokumen petisi *online* di situs change.org dikarenakan algoritme *Agglomerative* sesuai pada contoh kasus petisi *online* tersebut.

Penelitian kedua merupakan penelitian dari Husni, Dwi dan Syarief (2015) yang berjudul *Clustering Dokumen Web (Berita) Bahasa Indonesia Menggunakan Algoritma K-Means*. Pada penelitian tersebut mengelompokkan dokumen web berupa berita berbahasa Indonesia dengan menggunakan Algoritme K-Means, dan menjelaskan bahwa dokumen web yang sudah terkelompokkan dengan baik, akan memberikan akurasi yang memuaskan dalam hal pencarian. Berdasarkan analisis terhadap hasil ujicoba, penelitian tersebut menghasilkan 2 kesimpulan penting, yaitu: (1) Dokumen berita berhasil dikelompokan secara otomatis sesuai dengan derajat kesamaan berita sehingga menjadi kelompok dokumen berita yang terstruktur dengan diperoleh nilai rata-rata *F-Measure* 0.6129. (2) Jumlah *cluster*

dengan nilai *puritas* terbaik 0.75475 adalah 2 *cluster*. Akurasi yang masih belum sempurna dapat diperbaiki dengan memperbaiki teknik *preprocessing* dan melakukan analisis mengenai keakuratan dari metode lain, serta perlunya riset lanjutan untuk menentukan nilai *k* dengan lebih banyak proses percobaan pengukuran tingkat akurasi dan kemurnian *cluster*.

Penelitian ketiga merupakan penelitian dari Muflikhah dan Baharudin (2009) yang berjudul *High Performance in Minimizing of Term-Document Matrix Representation for Document Clustering*. Pada penelitian tersebut menjelaskan bahwa pengelompokan dokumen biasanya akan melibatkan data yang berdimensi tinggi. Penelitian tersebut menawarkan metode *Latent Semantic Indexing* (LSI) untuk proses pengelompokan dokumen dengan melibatkan *Singular Value Decomposition* (SVD) sehingga dapat mengidentifikasi *term* dan dokumen yang memiliki kemiripan tinggi. Pada proses penelitian tersebut, metode reduksi matrik dengan menggunakan berbagai jumlah pola (*rank*) dari SVD diaplikasikan pada dokumen *cluster* menggunakan algoritme *Fuzzy C-Means*. Hasil percobaan tersebut menghasilkan bahwa dokumen menjadi lebih baik dari sisi performansi bila diterapkan metode LSI daripada tanpa menggunakan metode LSI.

2.2 Petisi

Petisi merupakan sebuah dokumen tertulis resmi yang disampaikan kepada pihak berwenang untuk mendapatkan persetujuan dari pihak tersebut. Biasanya, hal ini ditandatangani oleh beberapa orang, menunjukkan bahwa sekelompok besar orang mendukung permintaan yang terdapat dalam dokumen. Di beberapa negara, hak masyarakat untuk mengajukan petisi dilindungi oleh hukum. Petisi juga merupakan salah satu bentuk strategi kampanye yang meliputi edukasi dan mobilisasi masyarakat terhadap permasalahan publik. Di mana edukasi publik dan mobilisasi cenderung mengarah pada bentuk yang menunjukkan dukungan masyarakat atas isu atau persoalan tertentu seperti petisi. Masyarakat menggunakan petisi untuk menarik perhatian pemerintah, dan sering berhasil karena dengan petisi sering kali memperoleh perhatian media. (Mustikaningsih, 2016).

2.3 Change.org

Change.org merupakan *website* yang sering digunakan oleh masyarakat dunia untuk menyampaikan petisi dan juga kampanye sosial secara *online*. Change.org juga bisa membuat setiap orang di mana saja memulai kampanye, memobilisai pendukung, dan bekerja dengan pengambil keputusan untuk mencari solusi. Change.org telah membuka kantor perwakilannya di indonesia dan juga telah menyediakan bahasa indonesia pada situs *website* tersebut. Situs ini telah ada di indonesia lebih dari 5 tahun. Sejak awal mula 2012 didirikan di indonesia, jumlah pengguna telah tumbuh dari 10 ribu sampai lebih dari 3,5 juta pengguna di seluruh dunia. Change.org telah banyak memenangkan petisi-petisi yang berisikan tentang penyelamatkan hutan, pemulihan hak suara, pembelaan hak penyandang cacat bahkan sampai melindungi satwa liar (Change.org inc, 2017).

2.4 Data Mining

Data mining adalah proses untuk menganalisa data pada data berjumlah besar atau berdimensi tinggi, *data mining* mengekstrak pola dan informasi yang berguna di dalamnya (Gullo,2015). Pada saat ini, *data mining* sudah banyak mendapat sebutan sebagai *business intelligence, knowledge discovery, predictive modelling , predictive analytics* dan sebutan lainnya. Tetapi pada umumnya tidak sedikit orang mendefinisikan *data mining* sebagai sinonim dari istilah popular lainnya yaitu *knowledge discovery from data* atau KDD dan yang lain sebagaimana *data mining* hanya sebagai salah satu tahap dari *knowledge discovery* (Jiawei, Kamber, dan Pei, 2012).

2.5 Text Mining

Text mining adalah teknik yang dapat digunakan dalam pengaplikasian metode klasifikasi maupun *clustering. Text mining* juga merupakan bagian dari *data mining* yang mana merupakan metode untuk menemukan pola-pola unik dari banyaknya data tekstual yang berjumlah besar atau berdimensi tinggi. Selain dapat digunakan sebagai klasifikasi dan *clustering*, *text mining* juga dapat digunakan dalam menyelesaikan beberapai persoalan lain seperti *information retrieval* dan *information extraction*.

Beberapa tahun terakhir, penelitian dan penggunaan *text* mining telah mendapat perhatian yang cukup besar, seiring dengan banyaknya data teks yang diperoleh dari berbagai *web*, dan jaringan sosial yang terhubung melalui internet. Sebagian besar data *text* tersebut berupa artikel berita, makalah, buku, blog, halaman *website*, dll. *Text mining* dapat menganalisis dokumen, meringkas dokumen, hingga mengelompokan dokumen berdasarkan kata-kata yang terkandung di dalamnya, serta dapat mencari kesamaan antar dokumen guna mengetahui hubungan dokumen tersebut dengan variabel lainnya. Contoh lain penerapan *text mining* adalah untuk penyaringan email spam, *analisis sentiment* dan pengelompokan topik-topik penelitian.

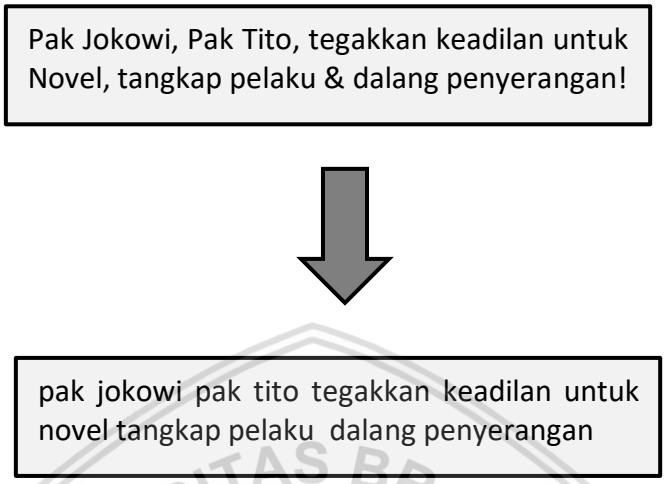
2.6 Text Pre-processing

Data yang di masukkan pada awal proses memerlukan *text preprocessing* terlebih dahulu supaya dapat dimengerti oleh sistem pengolahan *text mining* dengan baik. *Text preprocessing* adalah proses yang penting untuk menentukan kualitas pada proses selanjutnya seperti klasifikasi maupun pengelompokan *text*. Tujuan dari *text preprocessing* adalah untuk mendapatkan data yang sesuai pada proses *text mining* dari data awal yang masih berupa data tekstual. Beberapa tahapan dari *text preprocessing* terdiri dari *case folding, Tokenizing, Filtering* dan *Stemming*.

2.6.1 *Case Folding* dan *Tokenizing*

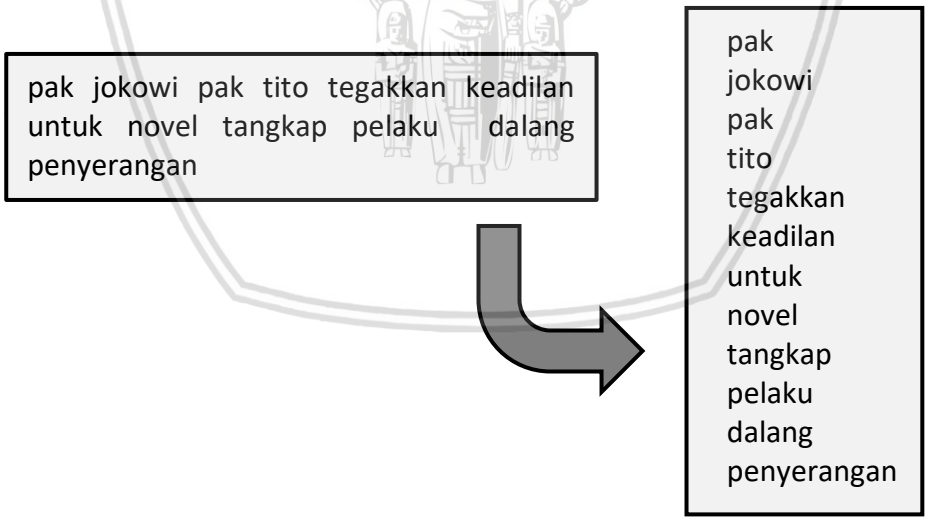
Pada awal *preprocessing* dokumen yang harus dilakukan adalah mengubah dokumen *input* secara *case folding*. *Case folding* merupakan proses mengubah

semua huruf pada *input* menjadi huruf kecil (Manning, Raghavan dan Schütze, 2009). Proses case folding juga akan menghapus tanda baca selain huruf '*a*' sampai '*z*'. Gambar 2.1 merupakan contoh tahap *case folding*.



Gambar 2.1 Case folding

Sedangkan *Tokenizing* merupakan proses pemotongan kalimat dalam setiap dokumen *input* menjadi satu kata atau per token. Sehingga pemrosesan selanjutnya akan lebih mudah dilakukan karena pemrosesan merupakan perkata. Gambar 2.2 merupakan proses dari *Tokenizing*.



Gambar 2.2 *Tokenizing*

2.6.2 Filtering

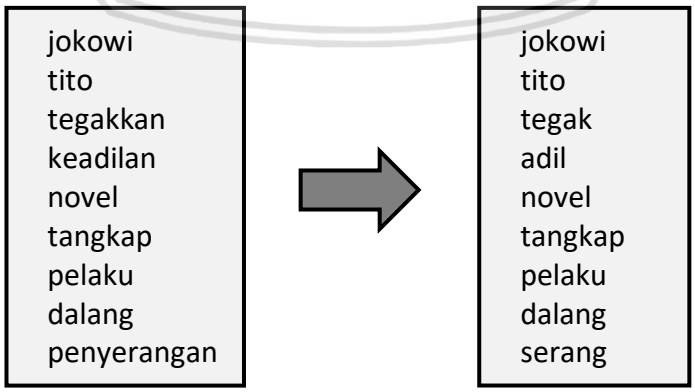
Filtering adalah proses penghapusan *term* atau kata-kata yang tidak memiliki makna, penghapusan kata yang tidak penting tersebut harus didasarkan pada sebuah *stopwords*. Kata-kata yang terdapat pada daftar *stopwords* harus dihilangkan (Lestari, Darma Putra dan Cahyawan, 2013).

Stopwords adalah kumpulan dari kata-kata yang tidak penting. Pembuangkan kata-kata pada dokumen dilakukan dengan pendekatan *bag-of-words* di mana struktur dari kalimat tidak diperhatikan. Pada pendekatan *bag-of-words*, kata-kata yang muncul pada *stopwords* akan dihapuskan. Gambar 2.3 merupakan contoh proses *Filtering*.

Gambar 2.3 *Filtering*

2.6.3 *Stemming*

Stemming adalah tahapan dari *Text Preprocessing* dokumen pada temu kembali informasi. *Stemming* mentransformasi kata-kata pada dokumen menjadi kata akarnya (*root word*) atau kata dasar. Perubahan kata menjadi kata dasar dilakukan dengan menghilangkan imbuhan pada kata tersebut. Proses tersebut banyak digunakan dalam *information retrieval* sebagai cara untuk meningkatkan performansi proses *retrieval* dokumen. Pemrosesan Dokumen yang telah mengalami proses *Stemming* dapat mengurangi ukuran file pada saat pengindekan (Tala, 2003). Gambar 2.4 merupakan contoh proses *Stemming*.



Gambar 2.4 *Stemming*

Terdapat beberapa algoritme *Stemming* yang dapat digunakan, salah satunya adalah *Stemming* Porter dan *Stemming* Nazief Adriani untuk *Stemming* berbahasa

indonesia. Penelitian ini algoritme *Stemming* yang digunakan adalah algoritme *Stemming* nazief dan adriani. Hal tersebut dikarenakan berdasarkan penelitian yang dilakukan oleh Agusta (2009), menyimpulkan bahwa Algoritme Nazief dan Adriani memiliki keakuratan (presisi) yang lebih besar dibandingkan dengan *Stemming* menggunakan algoritme porter.

Alur proses dari algoritme Nazief dan Adriani adalah:

1. Mencari kata *input* yang akan diproses pada kamus. Jika kata *input* sudah terdapat pada kamus, maka diasumsikan kata tersebut sudah berbentuk kata dasar dan proses algoritme diberhentikan.
2. Menghapuskan *Inflection suffixes* atau imbuhan ("-lah", "-kah", "-mu", "-ku", atau "-nya"). Jika berupa *particles* ("-lah", "-kah", "-tah" atau "-pun") maka langkah ini diproses lagi untuk menghapus *Possesive Pronouns* ("-mu", "-ku", atau "-nya"), jika ada.
3. Mengahapuskan *Derivation Suffixes* ("-an", "-i", atau "-kan"). Jika kata sudah terdapat dikamus , maka proses algoritme diberhentikan. Tetapi jika tidak maka ke langkah 3a.
    a. Jika "-an" sudah dihapus dan huruf terakhir dari kata tersebut adalah "-k", maka "-k" juga ikut dihapuskan. Jika kata terdapat pada kamus, maka proses algoritme diberhentikan. Tetapi jika tidak ditemukan pada kamus, maka dilanjutkan pada langkah 3b.
    b. Akhiran yang dihapus ("-an", "-i", atau "-kan") dikembalikan, dan lanjut ke langkah ke 4.
4. Menghapuskan *Derivation Prefix*. Jika pada langkah 3 ada sufiks yang dihapus maka ke langkah 4a, tetapi jika tidak maka ke langkah 4b.
    a. Periksa tabel kombinasi awalan-akhiran yang tidak diijinkan. Jika ditemukan maka algoritme berhenti, jika tidak maka menuju ke langkah 4b.
    b. For i = 1 *to* 3, tentukan tipe awalan kemudian hapus awalan. Jika *root word* belum juga ditemukan lakukan langkah 5, jika sudah maka algoritme berhenti. Catatan: jika awalan kedua sama dengan awalan pertama algoritme berhenti.
5. Melakukan *recoding*.
6. Jika ke 5 proses di atas sudah dilakukan tetapi tidak berhasil, maka proses dinyatakan selesai dan hasil dari proses *Stemming* merupakan kata awal *input* yang diasumsikan sebagai kata dasar (root word).

2.7 Term Weighting

Setelah dilakukan preprocessing maka proses selanjutnya adalah pembobotan kata (*Term Weighting*). Pembobotan kata adalah proses yang terjadi pada pengindekan teks untuk memberikan nilai pada setiap *term* yang terdapat pada dokumen. Pembobotan kata tersebut memberikan nilai numerik untuk istilah yang mewakili kepentingan suatu dokumen terhadap proses pengelompokan.

2.7.1 Pembobotan TFIDF

Pembobotan TFIDF merupakan metode untuk pembobotan yang paling umum digunakan untuk menggambarkan dokumen ke dalam model ruang vektor. TFIDF dapat digunakan pada permasalahan *information retrieval.* TFIDF juga di kenal efisien, mudah dan memiliki hasil yang akurat. TFIDF bisa diimplementasikan untuk pemodelan vektor klasifikasi teks ataupun pengelompokan teks. Metode pembobotan ini akan menghitung nilai *dari term frequency* (*TF*) dan juga menghitung nilai dari *Inverse Document Frequency* (*IDF*) pada setiap kata di dalam dokumen. *Term frequency* merupakan pembobotan dengan menghitung banyaknya kemunculan kata pada setiap dokumen. Persamaan 2.1 merupakan rumus perhitungan *term frequency*.

$$W_{TF\,(t,d)} = 1 + log(tf_{(t,d)}) \quad (2.1)$$

Keterangan:

$W_{TF_{(t,d)}}$: Pembobotan *term frequency*

$tf_{(t,d)}$: frekuensi kemunculan kata t pada suatu dokumen d

Sementara pada *inverse document frequency* (IDF) merupakan pembobotan yang mengukur seberapa pentingnya sebuah kata dalam dokumen dilihat dari keseluruhan dokumen secara global (Purwanti, 2015). Persamaan 2.2 merupakan rumus perhitungan pembobotan IDF.

$$W_{IDF\,(t,d)} = log\,\frac{N}{df_{(t,d)}} \quad (2.2)$$

Keterangan:

$W_{IDF\,(t,d)}$: pembobotan IDF (*Invers Document*)

N : banyaknya jumlah dokumen

$df_{(t,d)}$: banyaknya jumlah dokumen yang memiliki kata t pada dokumen

Setelah diketahui nilai dari $W_{TF\,(t,d)}$ dan juga $W_{IDF\,(t,d)}$ maka selanjutnya akan menghitung nilai dari TFIDF. Persamaan 2.3 merupakan rumus perhitungan TFIDF.

$$TFIDF_{(t,d)} = W_{TF\ (t,d)} \times W_{IDF\ (t,d)} \quad (2.3)$$

Keterangan:

$TFIDF_{(t,d)}$: pembobotan dokumen ke dalam model ruang vektor

$W_{TF\ (t,d)}$: Pembobotan TF (*term frequency*)

$W_{IDF\ (t,d)}$: pembobotan IDF (*Invers Document Frequency*)

Setelah melewati skema TFIDF, akan didapatkan hasil yang berupa matriks. Matriks yang didapatkan adalah matriks yang merepresentasikan dokumen dalam baris dan *token-token* atau kata yang sudah dipisah-pisahkan dalam kolom. Walaupun sudah memiliki bentuk yang sudah sesuai dan mampu diolah lebih lanjut menggunakan algoritme pembelajaran, tetapi hasil matriks yang didapatkan masih memiliki dimensi yang sangat tinggi.

2.8 Singular Value Decomposition

Singular Value Decomposition atau SVD digunakan untuk menurunkan dimensi pada matrik (reduksi matrik). *Singular Value Decomposition* adalah metode yang dapat digunakan pada reduksi matrik dengan cara menemukan korelasi dan struktur tersebunyi pada matrik. Pada dasarnya, metode SVD menggunakan dasar dari teorema aljabar linear, di mana matrik A akan diuraikan atau dipecah menjadi 3 *sub* matrik yang terdiri dari matrik *ortoghonal* U, matrik diagonal S dan *transpose* matrik *ortoghonal* V (Baker, 2013). Persamaan 2.4 merupakan rumus perhitungan SVD.

$$A_{mn} = U_{mn} S_{mn} V_{mn}^{T} \quad (2.4)$$

Keterangan:

A_{mn} : Matriks yang mewakili *m* jumlah dokumen dan *n* jumlah kata pada dokumen

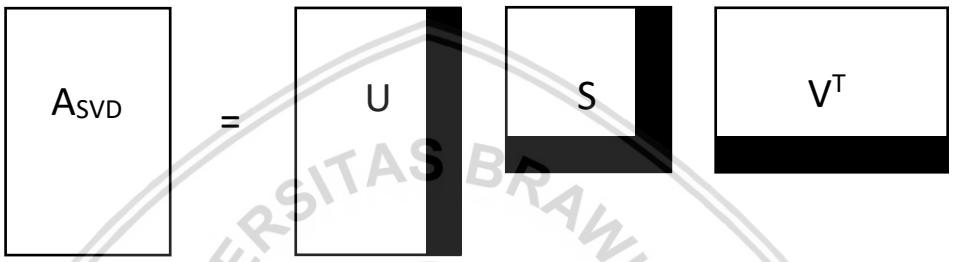
U : Vektor *eigen* ortonormal dari AA^T

V : Vektor *eigen* ortonormal dari $A^T A$

S : Matriks diagonal di mana nilai diagonalnya merupakan akar dari nilai U dan V yang disusun dengan urutan menurun bedasarkan besarnya nilai (nilai *singular* dari A)

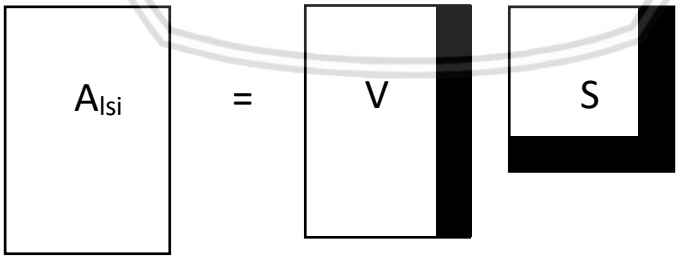
2.9 *Latent Semantic Indexing* (LSI)

Latent Semantic Indexing (LSI) adalah suatu metode untuk mendapatkan pola dalam suatu koleksi dokumen. LSI dapat memperbaiki akurasi dalam penerapan *information retrieval (IR).* Pada dasarnya, LSI menggunakan metode *Singular Value Decomposition* (SVD) untuk mendekomposisikan suatu matriks *term*-dokumen. LSI bisa digambarkan bahwa ukuran matrik yang diproses pada IR akan berkurang. Dengan berkurangnya ruang *term* dan dokumen menjadi dimensi yang lebih kecil, SVD memperlihatkan hubungan yang mendasari *term* dan dokumen dalam semua kombinasi yang memungkinkan dan membuang *noise* yang ada pada ruang vektor. (Muflikhah dan Baharudin, 2009). Matrik hasil SVD akan dipotong seperti pada gambar 2.5.



Gambar 2.5 Pemotongan matrik pada SVD

Pada matrik *Latent Semantic Indexing*, matrik hasil SVD yang digunakan adalah matrik V dan matrik S. Kedua matrik tersebut akan dikalikan dan dapat digunakan untuk membandingkan jarak dokumen satu dengan dokumen lainnya pada ruang vektor (Geib, 2011). Gambar 2.6 merupakan matrik yang dipotong pada pemrosesan *Latent Semantic Indexing*. Persamaan 2.5 merupakan rumus perhitungan LSI



Gambar 2.6 Pemotongan matik LSI

$$A_{lsi} = V_{mn} S_{mn} \tag{2.5}$$

Keterangan:

A_{lsi} : Matriks hasil reduksi matrik pada LSI

V_{mn} : Vektor *eigen* ortonormaI dari $A^T A$

S_{mn} : Matriks diagonaI di mana niIai diagonaInya merupakan akar dari nilai U dan V yang disusun dengan urutan menurun berdasarkan besarnya niIai (niIai *singuIar* dari A)

2.10 *Vector Space Model* (VSM)

Vector Space Model (VSM) telah banyak digunakan pada pemrosesan dokumen pada *text mining*. VSM merupakan model yang digunakan untuk mengukur *similarity* atau kemiripan antar dokumen. VSM mengubah koleksi dokumen ke dalam bentuk matrik. Operasi yang dilakukan pada model ruang vektor adalah *dot product*. Beberapa dokumen bisa diartikan sebagai sekumpulan vektor pada ruang vektor, yang berada pada satu sumbu untuk setiap *term*. Untuk mengukur kemiripan antar dokumen terdapat rumus *similarity* yang digunakan, adapun rumus yang digunakan adalah *cosine similarity*. Persamaan 2.6 merupakan rumus perhitungan *cosine similarity*.

$$cos\ \theta = \frac{\sum_{i=1}^{n} W_{i,j} W_{i,q}}{\sqrt{\sum_{i=1}^{n} W_{i,j}^2} \sqrt{\sum_{i=1}^{n} W_{i,q}^2}} \quad (2.6)$$

Keterangan:

$cos\ \theta$: nilai kemiripan antar dokumen

$W_{i,j}$: nilai bobot *term* ke– *i* pada dokumen ke– *j*

$W_{i,q}$: nilai bobot *term* ke- *i* pada dokumen ke – *q*

2.11 *Clustering*

Clustering adalah proses pengelompokan satu set data *object* menjadi beberapa kelompok atau *cluster* sehingga *object* dalam sebuah *cluster* memiliki kemiripan yang tinggi satu sama lain, tetapi sangat berbeda dengan *object* dalam kelompok lainnya.

Peng*cluster*an atau *Clustering* memiliki 2 metode dasar, yaitu *Hierarchical Clustering Method* dan *Non Hierarchical clustering Method*. *Hierarchical Clustering Method* digunakan apabila belum ditentukannya banyaknya *cluster* yang akan dihasilkan. Sedangkan non *hierarchical clustering* method adalah metode yang bertujuan untuk mengelompokkan sejumlah *n object* kedalam *k cluster*. Di mana nilai *k* < *n*.

2.11.1 UPGMA (*Unweighted Pair Group Method with Arithmetic Mean)*

Algoritme *Clustering* UPGMA (*Unweighted Pair-Group Method with Arithmetic Mean*) atau yang biasa disebut *Average Linkage* adalah algoritme pengelompokan untuk mengatasi permasalahan pada algoritme *Hierarchical clustering*. Adapun contoh lain dari *Hierarchical clustering* lainnya, yaitu *single linkage* dan *complete linkage*. Berbeda dengan kedua algoritme tersebut, UPGMA menggunakan pengukuran rata- rata dalam proses perhitungan jarak antar pasangan data yang telah di*cluster* dengan jarak data yang lainnya (Yim dan Ramdeen, 2015). Hasil keseluruhan dari algoritme UPGMA secara grafik dapat digambarkan sebagai *tree*, yang disebut dengan *dendrogram*.

Langkah-langkah pembentukan *cluster* algoritme UPGMA pada data dokumen.

1. Hitung jarak antar dokumen pada matriks dengan *cosine similarity*
2. Menentukan jarak minimum pada setiap dokumen pada matriks
3. Dokumen yang memiliki Jarak minimum digabungkan untuk membentuk *cluster* baru
4. Menghitung jarak antara *cluster* yang telah dibentuk pada langkah 3 dengan dokumen lainnya
5. Perhitungan jarak *cluster* baru dengan *cluster* lainnya diperoleh dengan rumus Persamaan 2.7

$$d_{UPGMA}A,B = \frac{\sum jarak\ antar\ dokumen}{m \times n} \quad (2.7)$$

Keterangan:

$d_{UPGMA}A,B$: jarak *cluster A* dan *B* dengan rata-rata kemiripan dokumen

m : jumlah data pada *cluster A*

n : jumlah data pada *cluster B*

6. Ulangi langkah 2 sampai 5 hingga semua dokumen tergabung menjadi 1 *cluster*
7. Gambar dendrogramnya

2.12 Evaluasi

Untuk menguji tingkat performansi dari algoritme *clustering* UPGMA, penelitian ini menggunakan beberapa pengukuran validitas untuk *clustering* dokumen. Analisis pengukuran performansi *cluster* ditentukan berdasarkan beberapa jumlah *cluster*, untuk menentukan berapa jumlah *cluster* yang terbaik, dan memiliki tingkat performansi tertinggi. Adapun pengujian yang dilakukan adalah.

2.12.1 Cophenetic Correlation Coefficient

Pengujian validitas pada *cluster* diperlukan untuk melihat kualitas (*quality*) hasil analisis pada *cluster*. Salah satu metode yang dapat digunakan adalah dengan menggunakan *Cophenetic Correlation Coefficient* (CPCC). *Cophenetic Correlation Coefficient* merupakan koefisien korelasi antara matrik jarak dan matrik jarak *dendrogram* atau matriks *cophenetic* (Saracli et al, 2013. *Cophenetic Correlation Coefficient* dapat dihitung dengan rumus Persamaan 2.8.

$$c = \frac{\sum_{i<j}(x(i,j)-x)(t(i,j)-t)}{\sqrt{[\sum_{i<j}(x(i,j)-x)^2][\sum_{i<j}(t(i,j)-t)^2]}} \quad (2.8)$$

Keterangan:

c : Koefisien korelasi *cophenetic*

$x(i,j)$: jarak dokumen *x* ke-*i* dan dokumen *x* ke-*j*

$t(i,j)$: jarak *cophenetic* dokumen *x* ke-*i* dan dokumen *x* ke-*j*

x : rata-rata jarak $x(i,j)$

t : rata-rata jarak *cophenetic* $t(i,j)$

2.12.2 *Silhouette Coefficient*

Silhouette Coefficient adalah metode yang digunakan untuk mengukur kualitas dan kekuatan suatu *cluster*, di mana pada *cluster* tersebut dapat diukur seberapa baik suatu *object* sudah ditempatkan pada *cluster* yang sesuai. *Silhouette Coefficient* merupakan gabungan dari metode *cohesion* dan *separation* (Wahyuni et al, 2016). Beberapa tahapan dalam menghitung *Silhouette Coefficient* dengan jarak *cosine similarity* yaitu.

1. Menghitung rata – rata jarak dari data yang terpilih menggunakan Persamaan 2.9.

$$a_i = \frac{1}{m_j-1}\sum_{r=1}^{m_j} d\left(x_i^j, x_r^j\right) \quad (2.9)$$

Keterangan:

a_i : jarak rata-rata *object i* dengan seluruh *object* yang berada dalam satu *cluster*

$d\left(x_i^j, x_r^j\right)$: jarak antar dokumen *x* ke-*i* dan dokumen *x* ke-*r*

Dan Persamaan 2.10.

$$b_i = max\left\{\frac{1}{m_n}\sum_{r=1}^{m_n} d\left(x_i^j, x_r^n\right)\right\} \quad (2.10)$$

Keterangan:

b_i : jarak rata-rata *object i* dengan seluruh *object* yang berada di *cluster* lainnya

$d\left(x_i^j, x_r^n\right)$: jarak antar dokumen *x* ke-*i* dan dokumen *x* ke-*r*

2. Kemudian dari Persamaan tersebut dapat menghitung Nilai S_i dengan Persamaan 2.11.

$$S_i = \begin{cases} 1 - \frac{b_i}{a_i} & jika\ a_i > b_i \\ 0 & jika\ a_i = b_i \\ \frac{a_i}{b_i} - 1 & jika\ a_i < b_i \end{cases} \quad (2.8)$$

Keterangan:

S_i : nilai *Silhouette Index*

a_i : jarak rata-rata *object i* dengan seluruh *object* yang berada dalam satu *cluster*

b_i : jarak rata-rata *object i* dengan seluruh *object* yang berada di *cluster* lainnya

Pada permasalahan *cluster* yang hanya memiliki satu *object*, maka nilai *silhouette* pada *object* tersebut di *set* dengan nilai 0. *Silhouette Index* merepresentasikan bentuk grafik yang mana akan menunjukkan seberapa mirip suatu *object* data yang merupakan sebuah *dataset* ke *object* lain yang sama.

BAB 3 METODOLOGI

Pada bab ini membahas metode-metode yang menjadi acuan dalam perancangan pengelompokan dokumen petisi pada situs change.org dengan algoritme *hierarchical clustering* UPGMA. Tahapan metodologi penelitian memiliki struktur yang dapat dilihat pada Gambar 3.1.

Gambar 3.1 Diagram Blok Metodologi Penelitian

3.1 Studi Pustaka

Studi pustaka yang digunakan dalam merancang sistem diperoleh melalui berbagai sumber informasi seperti *e-book*, berita *online*, jurnal, maupun informasi yang ada di internet. Hal-hal yang dilakukan untuk mendukung penelitian ini, diantaranya adalah pemahaman tentang *Data mining* dan *Text mining*. Penekanan yang menjurus pada *clustering* memuat tentang pemahaman *prepossessing* dokumen, penerapan TFIDF, reduksi *matrix*, *normalisasi*, *cosine similarity* dan evaluasi *clustering*.

3.2 Analisis Kebutuhan

Kebutuhan-kebutuhan dalam membangun sistem dijabarkan pada tahapan analisis kebutuhan ini. Hal-hal yang membantu proses pengelompokan dokumen petisi pada situs change.org dengan algoritme *hierarchical clustering* UPGMA:

1. Kebutuhan *Hardware*, meliputi:
   a. Laptop
2. Kebutuhan *Software*, meliputi:
   a. Sistem Operasi *Windows* 10
   b. Bahasa pemrograman *JAVA*
   c. Aplikasi Netbeans + GUI
3. Data yang dibutuhkan, meliputi:
   a. Data dokumen petisi dari situs change.org berbahasa indonesia

3.3 Pengumpulan Data

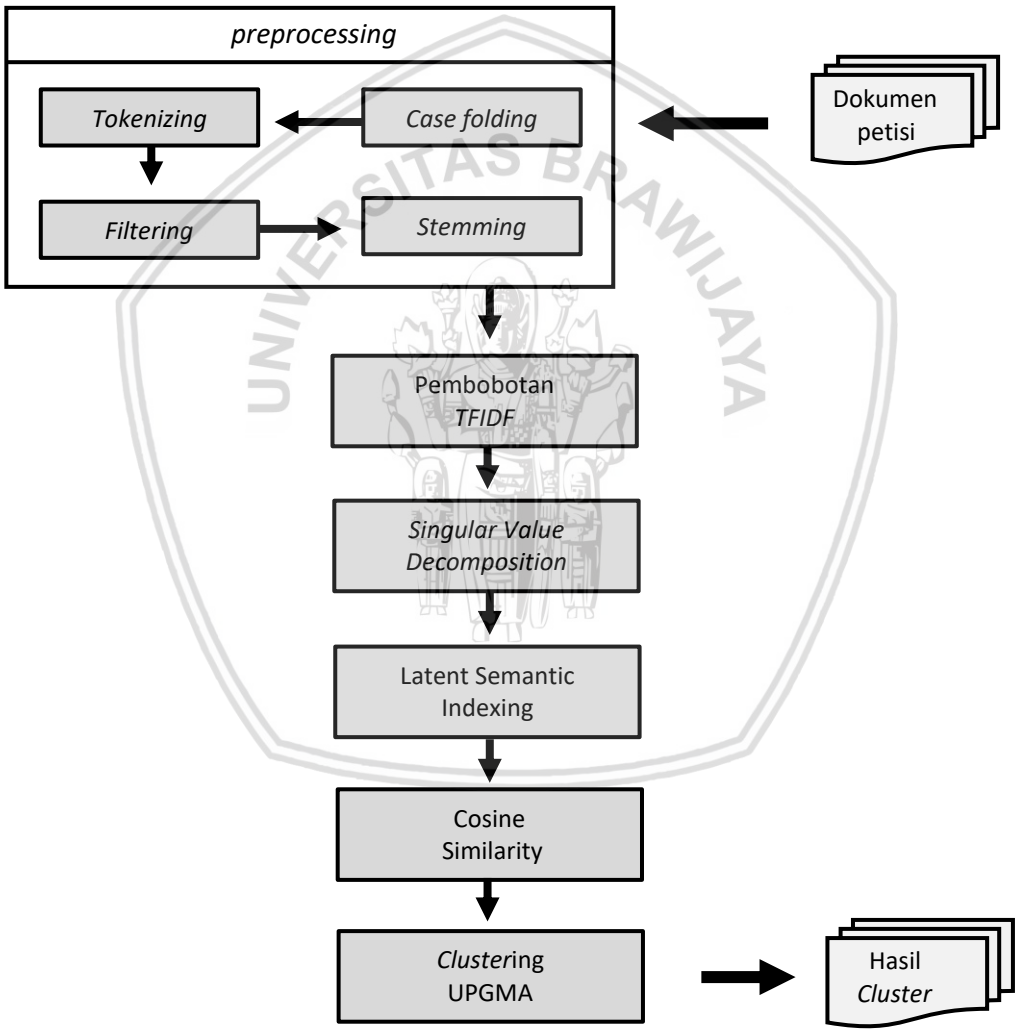
Pengumpulan data yang dilakukan merupakan data yang terdapat pada situs (*www.*change.org). Data petisi merupakan data yang berbahasa Indonesia dan dari kurun waktu antara Januari 2013 sampai dengan Juli 2017. Jumlah data yang digunakan adalah sebanyak 100 dokumen. Petisi *online* yang dijadikan sebagai *dataset* penelitian ini merupakan petisi yang diklasifikasikan populer pada situs tersebut. Petisi pada change.org yang populer di rasa memiliki tingkat kepercayaan juga kebenaran yang tinggi dikarenakan memiliki basis pendukung yang banyak. Proses pengumpulan data petisi pada situs change.org dilakukan secara manual dan data petisi yang dipilih dari fitur populer dipilih secara acak. Konten petisi yang dijadikan *dataset* adalah judul petisi dan isi konten petisi.

3.4 Perancangan Sistem

Perancangan sistem menjelaskan alur kerja keseluruhan dalam penelitian ini. Dengan adanya perancangan sistem diharapkan dapat mempermudah peneliti saat pengimplementasian sistem yang akan dibuat.

Pada proses perancangannya terdapat beberapa tahapan diantaranya pengumpulan *dataset* petisi *online* pada situs change.org dan disimpan pada file berformat *.txt. file yang terbaca akan dilakukan preprocessing *text* sehingga dokumen petisi akan dilakukan penyaringan berupa *case folding, Tokenizing, Filtering* dan *Stemming*. Pada proses *Filtering*, daftar *stopwords* yang digunakan

adalah daftar *stopwords* dari Tala (2003). Data hasil dari *preprocessing* berupa *term-term* yang telah berbentuk kata dasar. Selanjutnya akan dilakukan pembobotan pada setiap *term* dengan menggunakan pembobotan *TFIDF*. Kemudian dilakukan reduksi matrik untuk mengecilkan nilai dan mempercepat komputasi matrik dengan *Singular Value Decomposition* kemudian ditambahkan dengan metode *Latent Semantic Indexing* (LSI). Setelah dilakukan proses LSI, maka dilanjutkan dengan menghitung kedekatan setiap dokumen dengan *cosine similarity*. Hasil matrik inilah yang akan digunakan untuk menghitung *clustering* UPGMA (*Unweighted Pair Group Method with Arithmetic Mean)*, sehingga pada setiap dokumen yang sudah dimasukkan akan digolongkan menjadi beberapa *clustering* hingga berbentuk hirarki berjumlah satu *cluster*. Gambar 3.2 merupakan alur rancangan sistem.



Gambar 3.2 Diagram Perancangan Sistem

3.5 Implementasi Sistem

Setelah dilakukan perancangan sistem maka langkah selanjutnya melakukan implementasi kedalam program guna menghasilkan sistem yang dapat mengelompokkan petisi *online* pada situs change.org secara otomatis. Pembuatan

aplikasi *clustering* petisi *online* dilakukan dengan menggunakan bahasa pemrograman *JAVA* yang meliputi beberapa hal di bawah ini:

1. Pembuatan antarmuka menggunakan GUI pada *JAVA*.
2. *Input text* petisi *online* dengan format *.txt.
3. Melakukan *preprocessing* data petisi *online.*
4. Melakukan pembobotan dengan TFIDF.
5. Melakukan reduksi matrik guna mengecilkan nilai pada matrik.
6. Pengelompokkan dengan algoritme UPGMA *clustering.*
7. Hasil *output* berupa dokumen yang telah ter*cluster* secara hirarki.

3.6 Pengujian

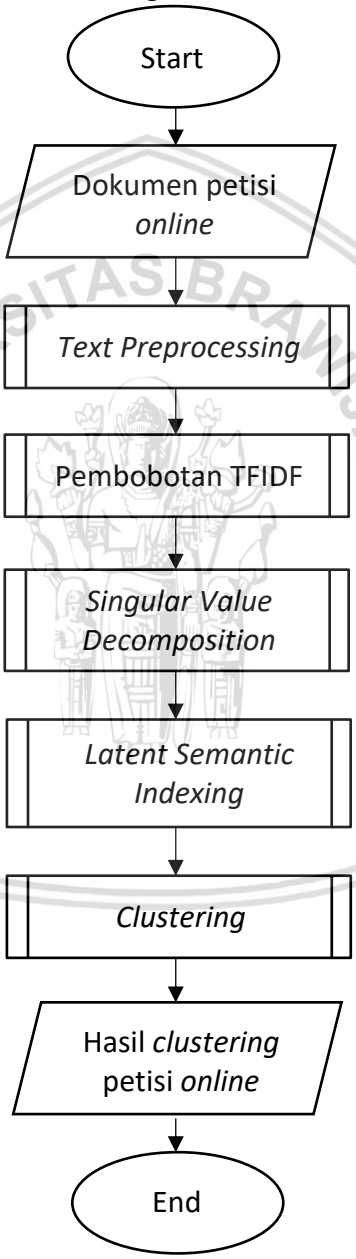
Pada pengujian akan dilakukan penelitian mengenai tingkat performansi pada hasil dari pengelompokan petisi *online* pada situs change.org. Performansi dinilai berdasarkan algoritme evaluasi yaitu *Cophenetic Correlation Coefficient* dan *Silhouette Coefficient*. Pada evaluasi tersebut akan bisa dinilai apakah hasil dari suatu data petisi sudah sesuai pada *cluster* yang dikelompokkan atau tidak. Evaluasi tersebut juga bisa menilai apakah algoritme *hierarchical clustering* UPGMA sudah sesuai untuk diimplementasikan pada *dataset* dokumen petisi *online* atau sebaliknya.

3.7 Kesimpulan

Setelah penjabaran mengenai latar belakang masalah petisi *online*, landasan pustaka yang membahas metode yang akan digunakan, juga metodologi penelitian, dapat disimpulkan bahwa petisi *online* pada situs change.org membutuhkan suatu analisis *clustering* untuk memudahkan pengguna dalam hal pencarian dan pengelolaan dokumen supaya tertata dan mudah dalam pengoperasiannya. Algoritme *clustering* UPGMA merupakan algoritme yang sesuai untuk diterapkan pada kasus pengelompokan petisi *online* tersebut.

BAB 4 PERANCANGAN

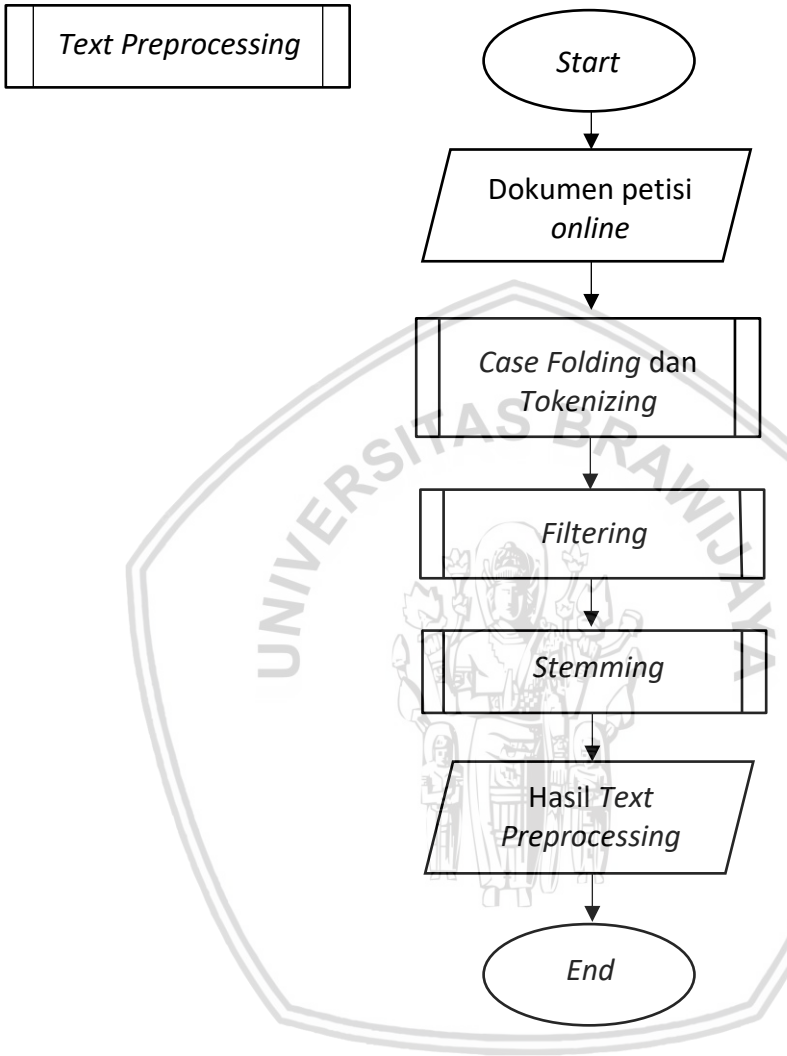
Pembangunan sistem pada penelitian secara umum bertujuan untuk mengelompokkan dokumen petisi *online* di situs change.org. Sistem memerlukan perancangan sistem yang berisikan proses-proses yang berkaitan pada pengelompokkan petisi *online*. Gambar 4.1 merupakan alur sistem terdiri dari 5 tahap yaitu *Text Preprocessing,* pembobotan TFIDF, *Singular Value Decomposition, Latent Semantic Indexing*, dan *clustering*.



Gambar 4.1 Diagram alir sistem

4.1 Text Preprocessing

Tahapan dari *Text Preprocessing* merupakan tahap awal dari pemrosesan *text* untuk mengolah *text* yang belum terstruktur menjadi lebih terstruktur. Gambar 4.2 merupakan diagram alir *Text Preprocessing*.

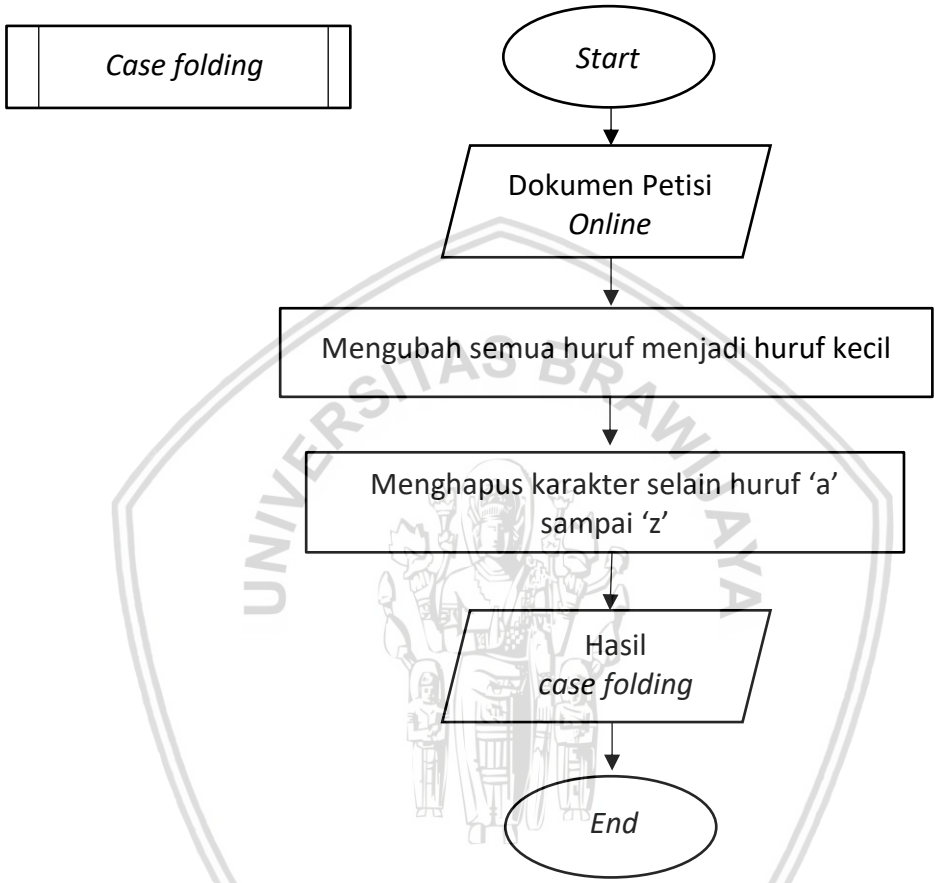


Gambar 4.2 Diagram alir *text preprocesssing*

Gambar 4.2 menjelaskan alur dari *Text Preprocessing* untuk menstrukturkan dokumen pada petisi *online*. Pada alur awal diagram, terdapat proses memasukkan data yang diperoleh dari situs change.org yaitu dokumen petisi *online*. Selanjutnya dokumen akan diproses dengan proses *case folding* untuk mengubah semua kata pada dokumen menjadi *lowercase* atau huruf kecil. Setelah itu dilanjutkan dengan proses *Tokenizing* dengan memecah dokumen berdasarkan kata atau per *token*. Setelah itu mengalami proses *Filtering* untuk menyaring kata atau menghapus kata yang terdapat pada *stopwords* yang masuk kedalam kata yang tidak penting sehingga dihapuskan pada kumpulan *token*. Setelah *Filtering* selesai dilanjutkan dengan proses pembentukan kata dasar pada proses *Stemming*.

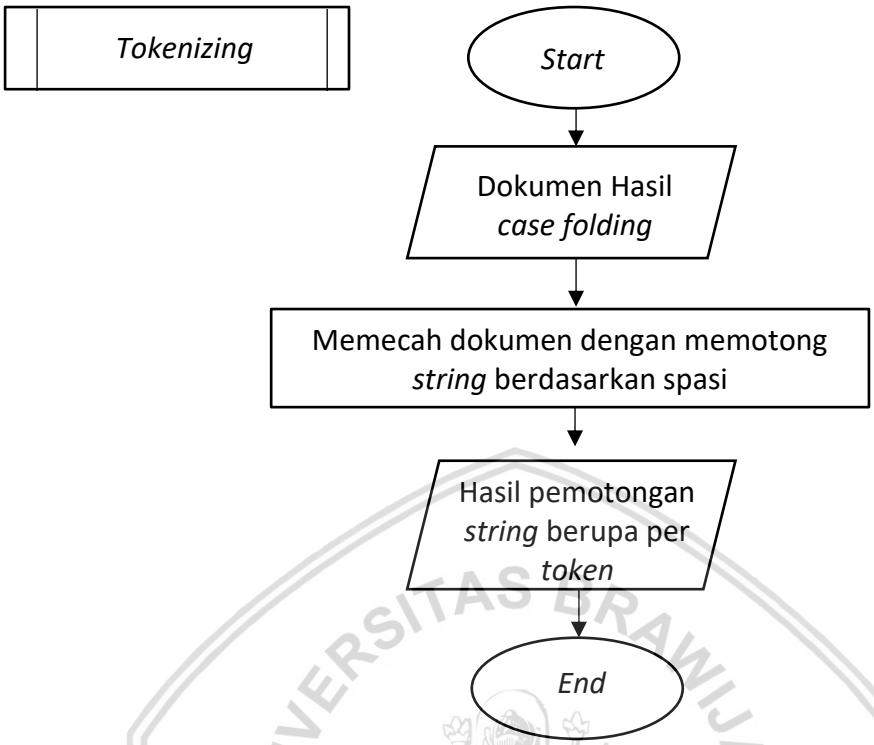
4.1.1 *Case Folding* dan *Tokenizing*

Case folding digunakan untuk mengubah semua huruf menjadi huruf kecil dan dan menghapus tanda baca selain hurus ‘a’ sampai ‘z’. Gambar 4.3 merupakan alur *case folding*.



Gambar 4.3 Diagram alir *case folding*

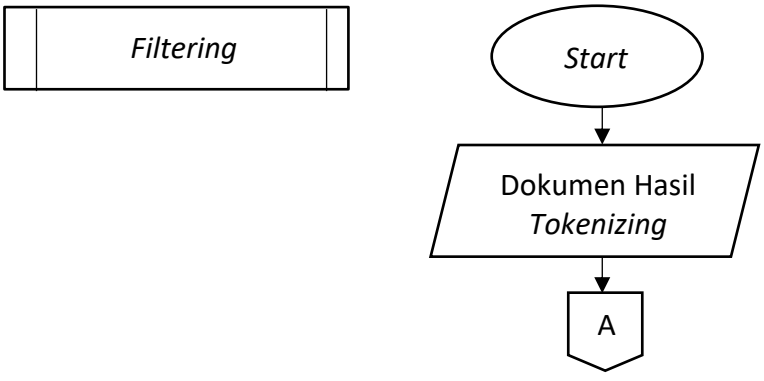
Sedangkan untuk *Tokenizing* merupakan proses untuk memotong kalimat menjadi satu kata atau per *token*. Gambar 4.4 merupakan diagram alir *Tokenizing*.

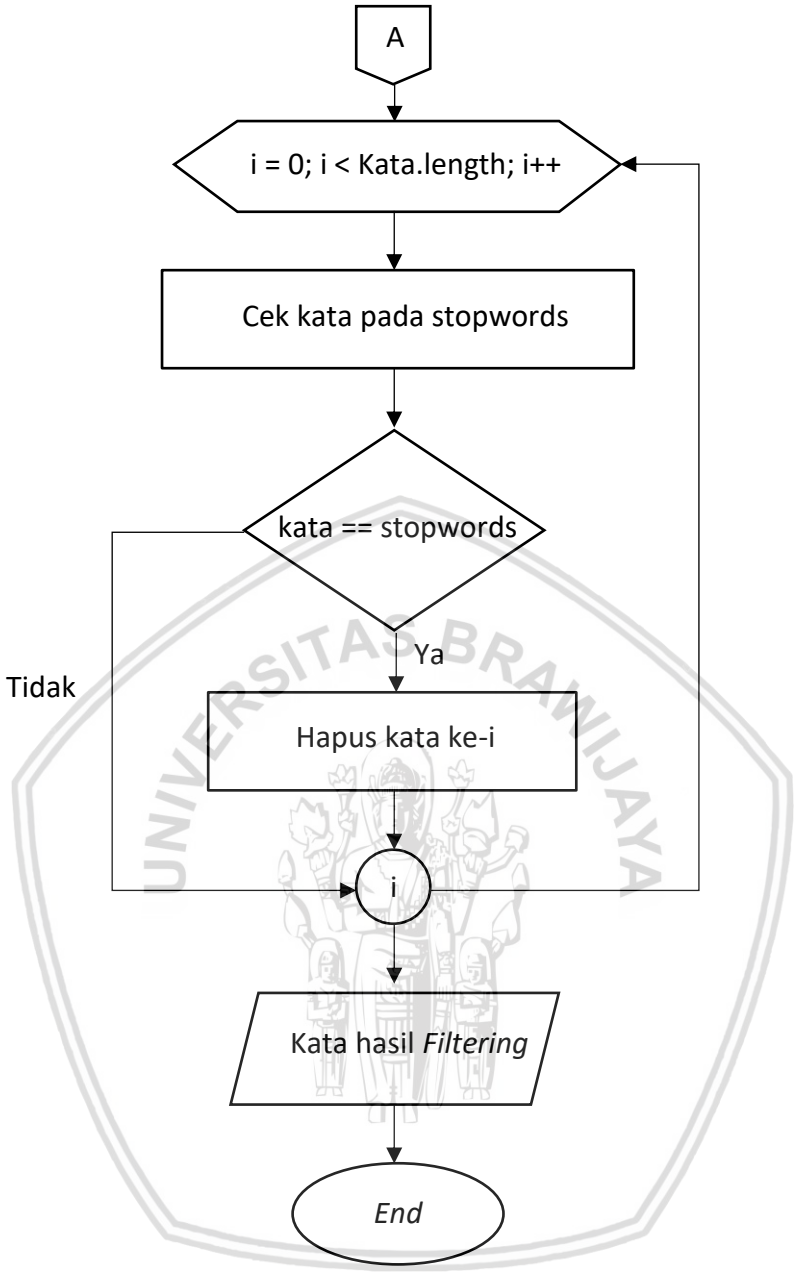


Gambar 4.4 Diagram alir *Tokenizing*

4.1.2 Filtering

Filtering merupakan *sub* bagian dari *Text Preprocessing*, pada *Filtering* dilakukan proses penghapusan kata-kata yang tidak memiliki makna, penghapusan kata yang tidak penting tersebut harus didasarkan pada sebuah *stopwords* atau kumpulan dari kata-kata yang tidak penting. Gambar 4.5 merupakan alur dari proses *Filtering*.

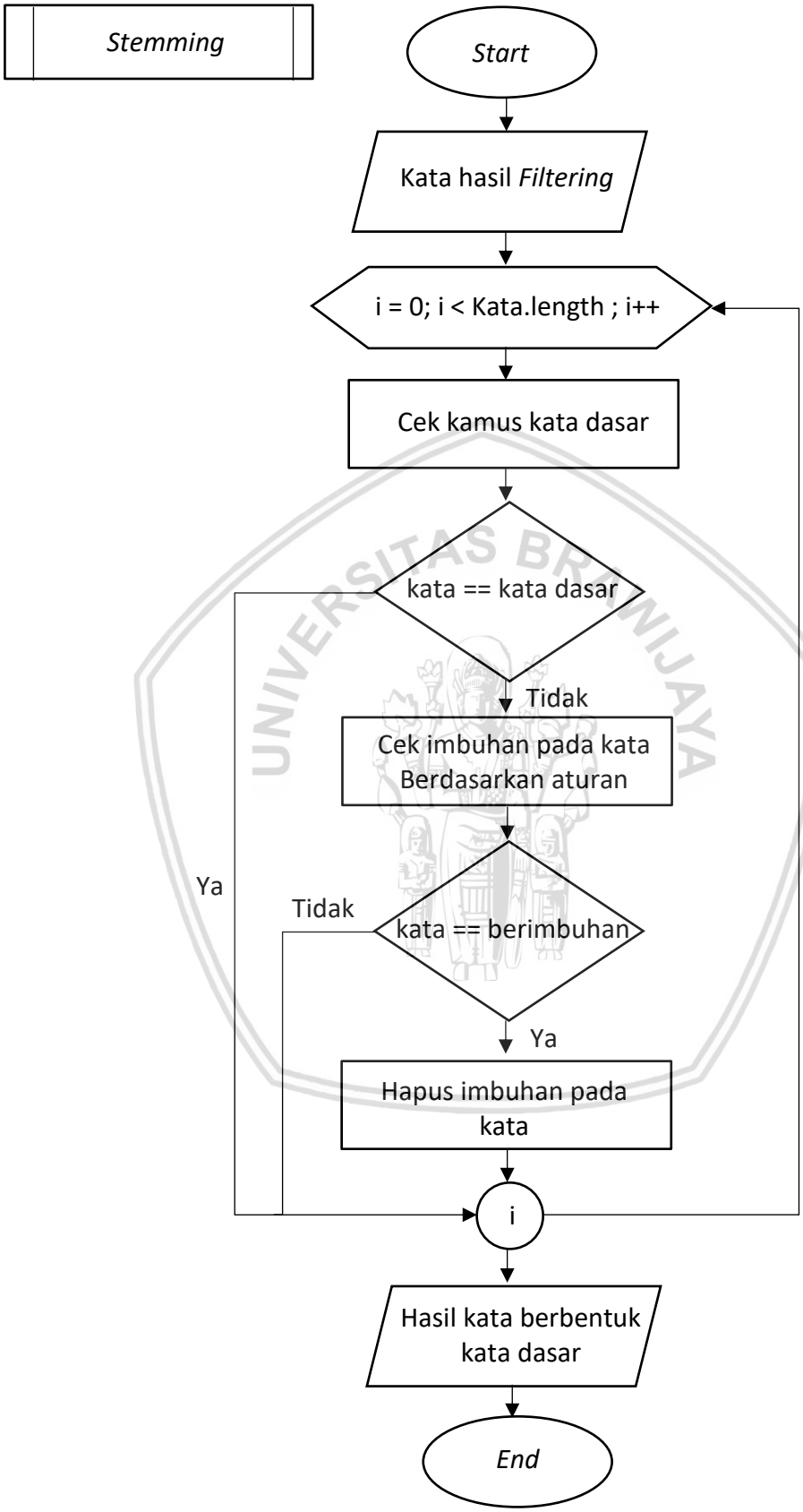




Gambar 4.5 Diagram alir *Filtering*

4.1.3 *Stemming*

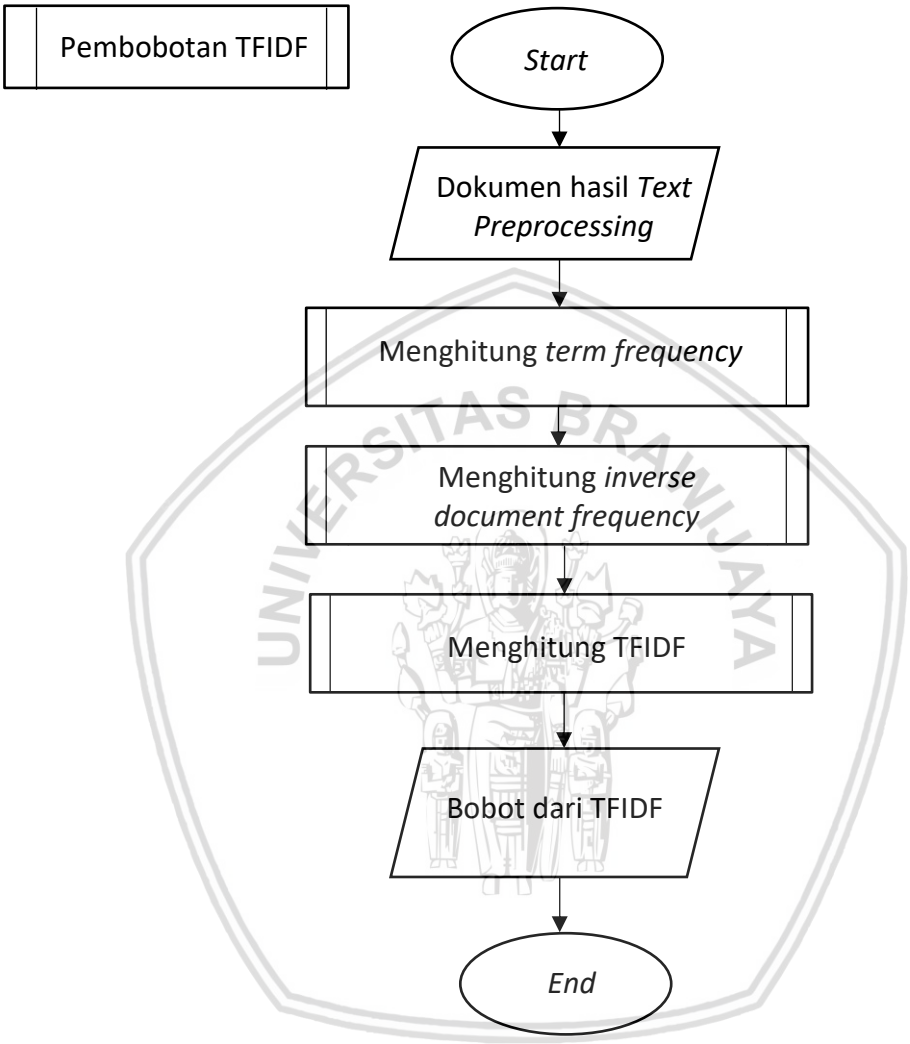
Stemming merupakan tahapan lanjutan setelah dilakukan proses *Filtering*. Pada proses ini *token* atau kata hasil dari *Filtering* akan ditransformasikan menjadi kata dasarnya (*root word*). Proses transformasi dilakukan dengan menghilangkan imbuhan pada kata tersebut. Gambar 4.6 merupakan diagram alir dari proses *Stemming*.



Gambar 4.6 Diagram alir *Stemming*

4.2 Pembobotan TFIDF

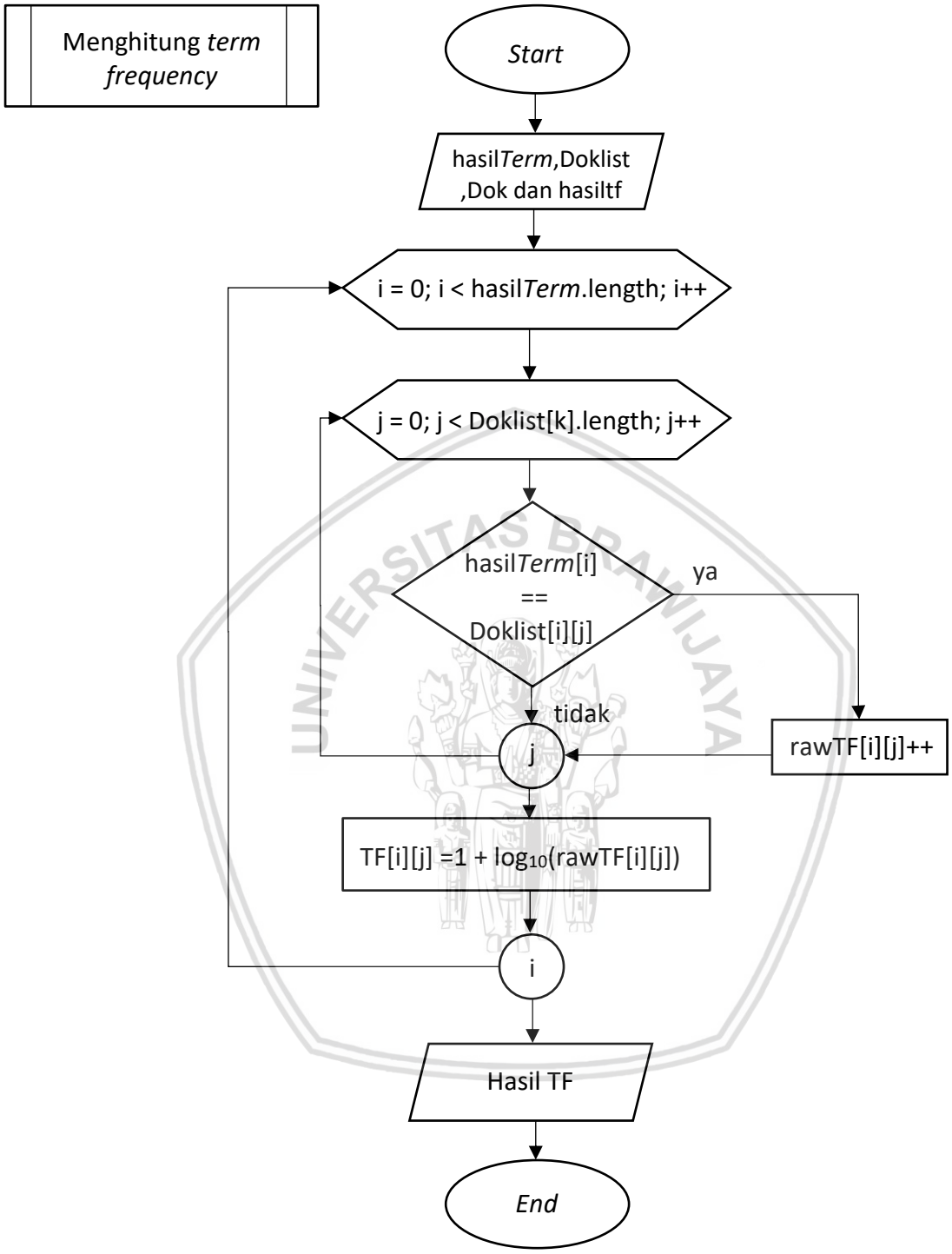
Setelah proses *Text Preprocessing* sudah dilakukan maka langkah setelahnya merupakan pembobotan TFIDF. Pada Pembobotan ini bertujuan untuk mengubah data yang awalnya berupa *text* menjadi data *numerik*. Gambar 4.7 Merupakan diagram alir proses pembobotan TFIDF.



Gambar 4.7 Diagram alir Pembobotan TFIDF

4.2.1 Menghitung *term* frequency

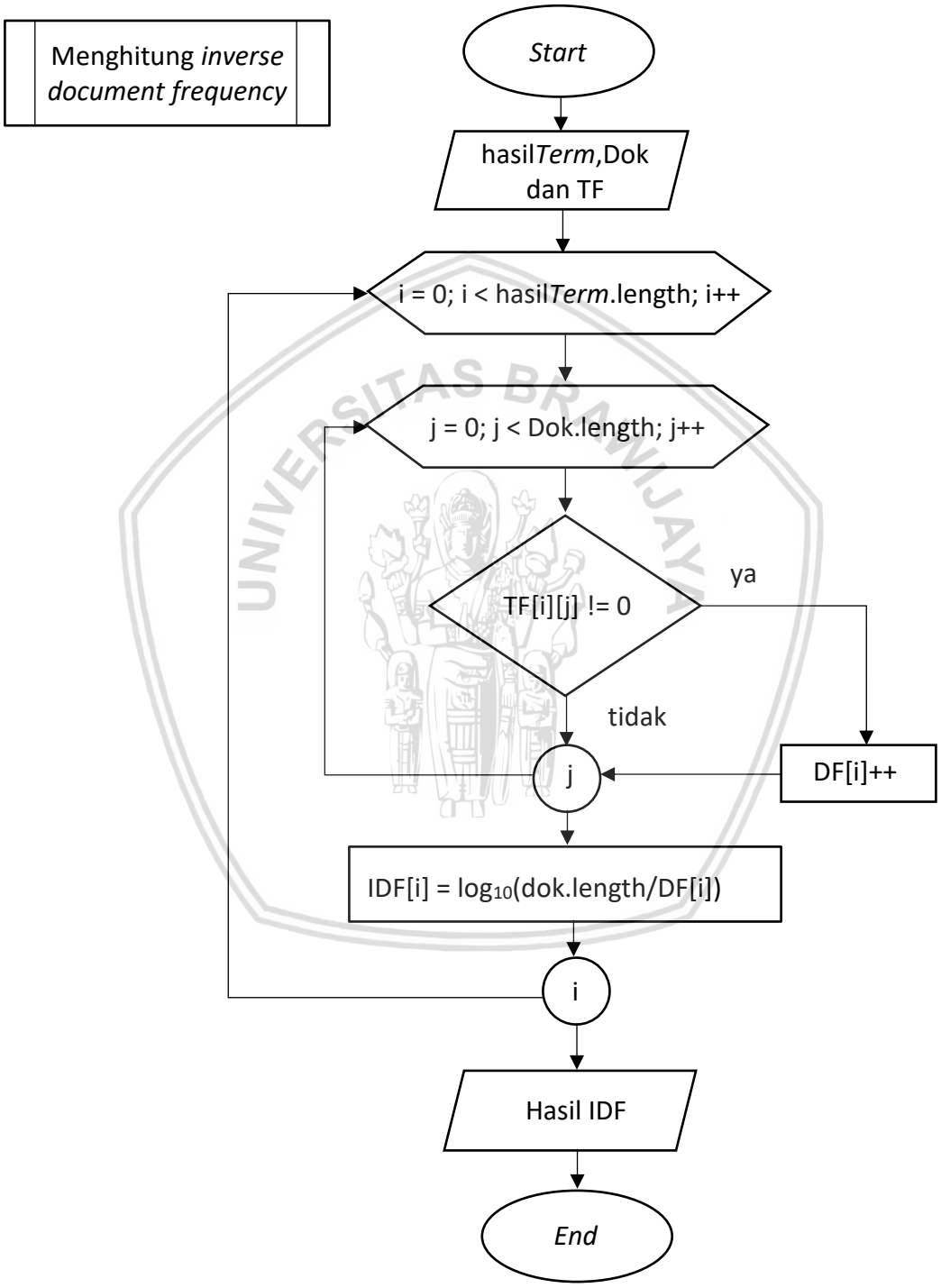
Proses *Term frequency* merupakan pembobotan dengan menghitung banyaknya kemunculan kata pada setiap dokumen. Gambar 4.8 Merupakan diagram alir proses perhitungan *term frequency.*



Gambar 4.8 Diagram alir perhitungan TF

4.2.2 Menghitung inverse document frequency

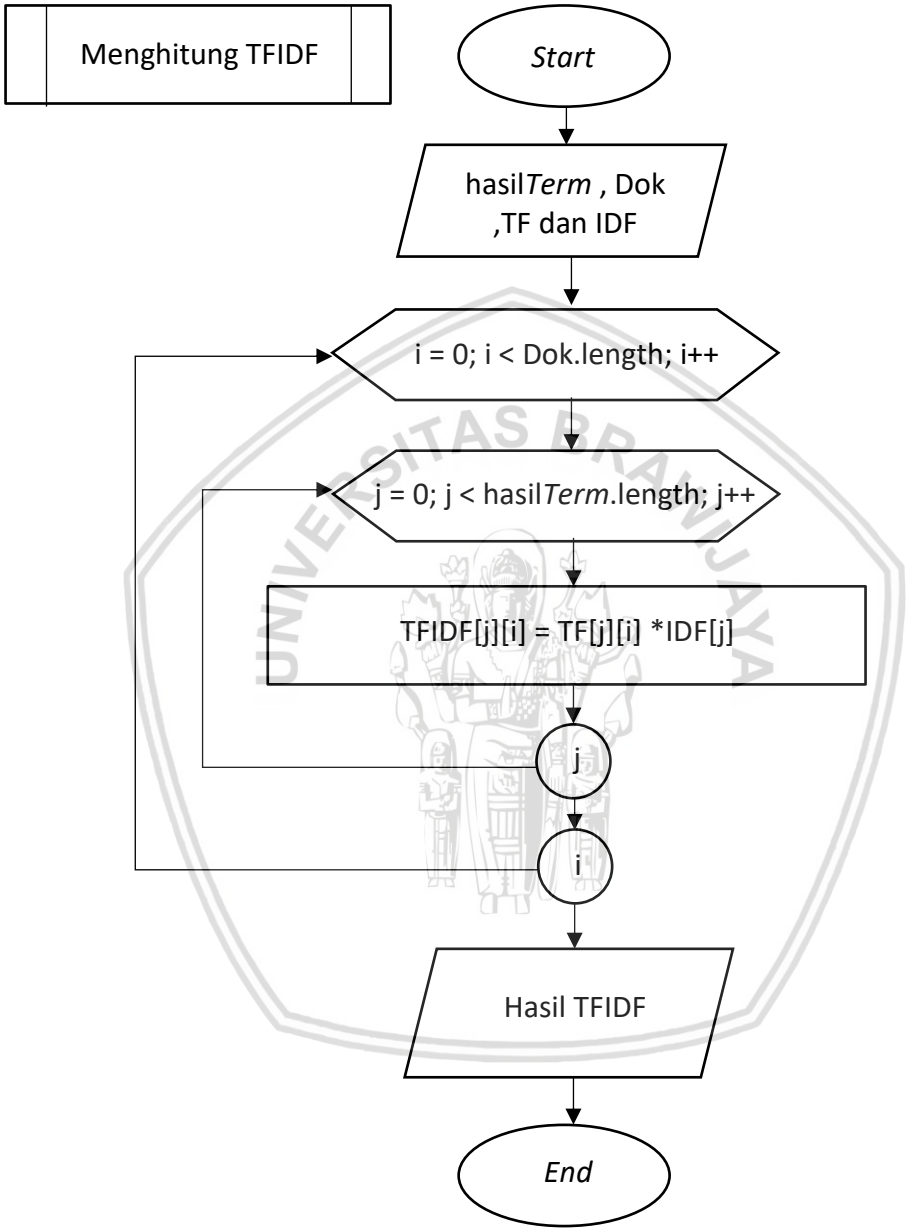
inverse document frequency (IDF) merupakan pembobotan yang mengukur seberapa pentingnya sebuah kata dalam dokumen dilihat dari keseluruhan dokumen secara global. Gambar 4.9 Merupakan diagram alir proses perhitungan IDF.



Gambar 4.9 Diagram alir perhitungan IDF

4.2.3 Menghitung TFIDF

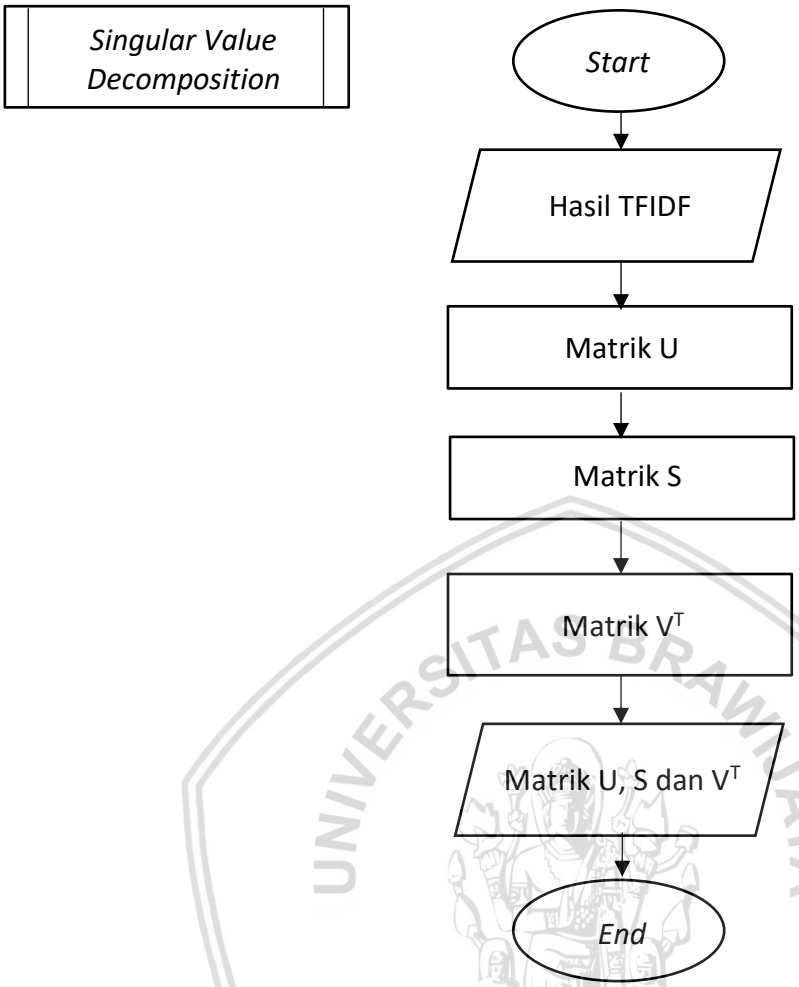
Pembobotan pada setiap *term* akan dihitung dengan menggunakan perhitungan TFIDF pada rumus Persamaan 2.3. Gambar 4.10 merupakan diagram alir perhitungan TFIDF.



Gambar 4.10 Diagram alir perhitungan TFIDF

4.3 Singular Value Decomposition

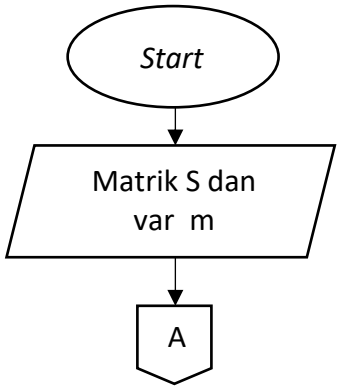
Pada *Singular Value Decomposition* matrik hasil pembobotan TFIDF akan dipecah menjadi 3 matrik, yaitu matrik U, matrik S dan matrik V *transpose*. Gambar 4.11 merupakan diagram alir proses *Singular Value Decomposition.*

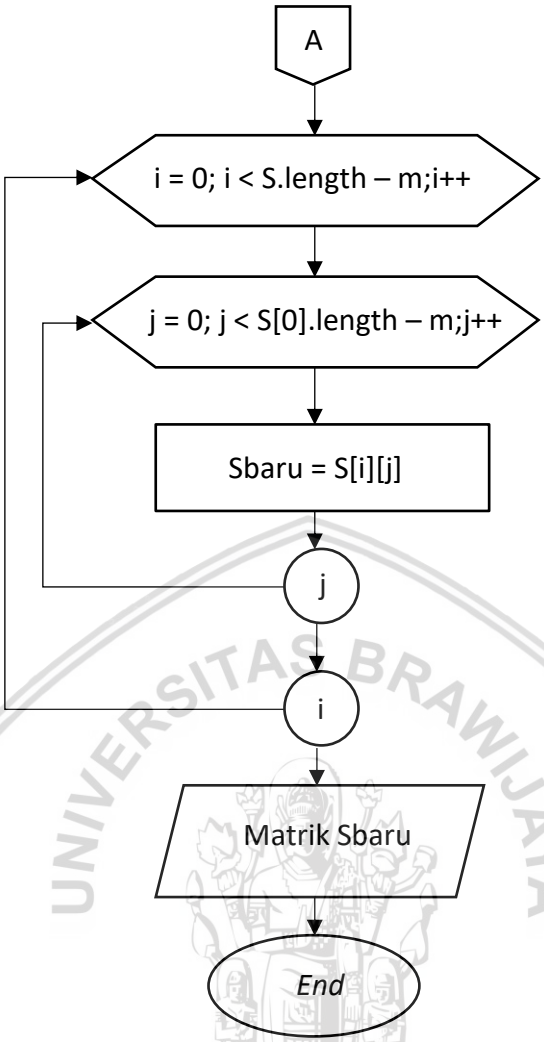


Gambar 4.11 Diagram alir *Singular Value Decomposition*

4.4 Latent Semantic Indexing

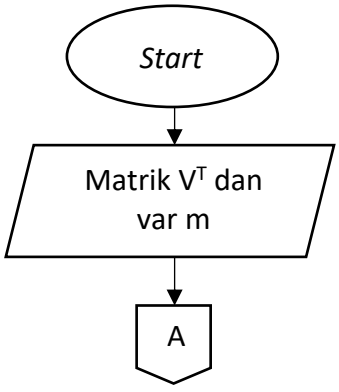
Latent Semantic Indexing digunakan untuk mereduksi matrik hasil pemecahan *Singular Value Decomposition*. Sebelum menggunakan rumus Persamaan 2.5. Matrik *S* dan matrik V^T hasil SVD akan dilakukan pemotongan sebanyak *m* baris dan *m* kolom untuk matrik *S* sedangkan matrik V^T dipotong sebanyak *m* baris. Gambar 4.12 merupakan diagram alir pemotongan matrik *S*.

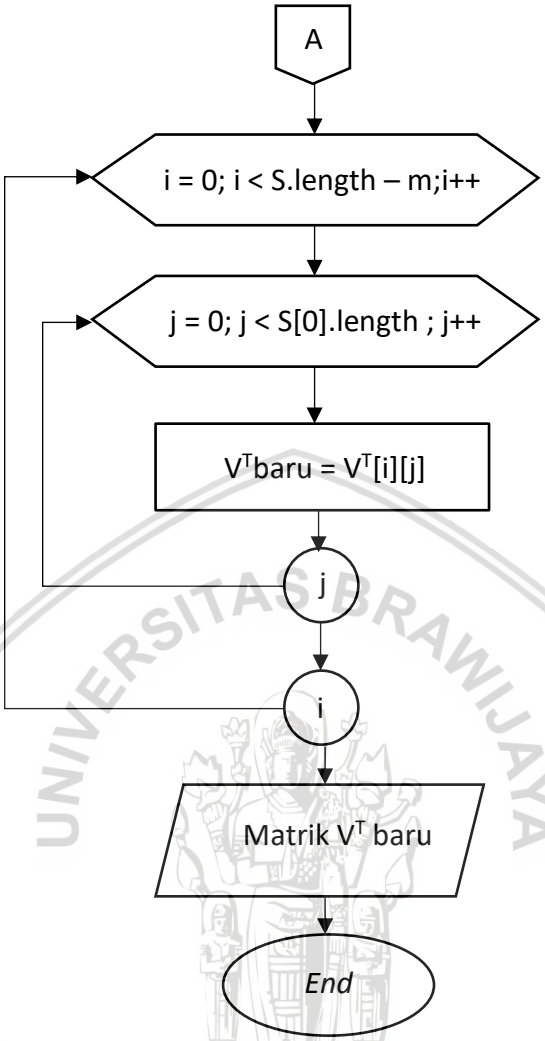




Gambar 4.12 Diagram pemotongan matrik S

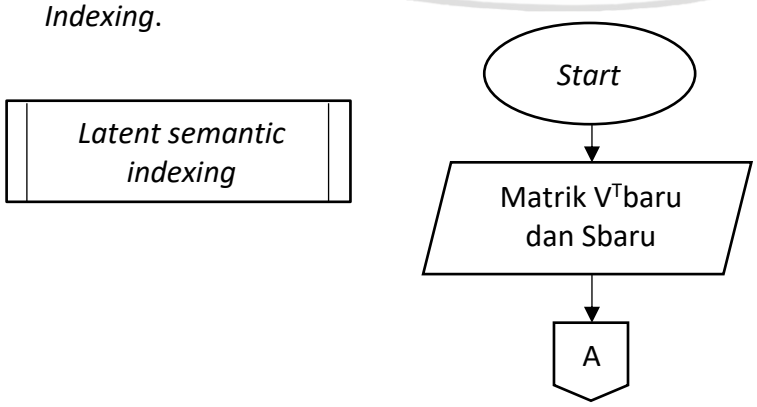
Selain pemotongan pada matrik *S* selanjutnya juga dilakukan pemotongan matrik pada matrik V^T. Gambar 4.13 merupakan diagram alir pemotongan matrik V^T.

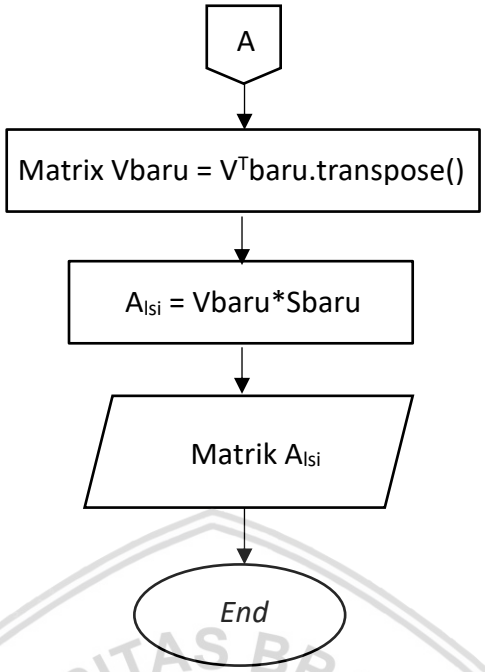




Gambar 4.13 Diagram pemotongan matrik V^T

Selanjutnya adalah proses perhitungan *Latent Semantic Indexing* pada Persamaan 2.5. Gambar 4.14 merupakan diagram alir proses *Latent Semantic Indexing*.

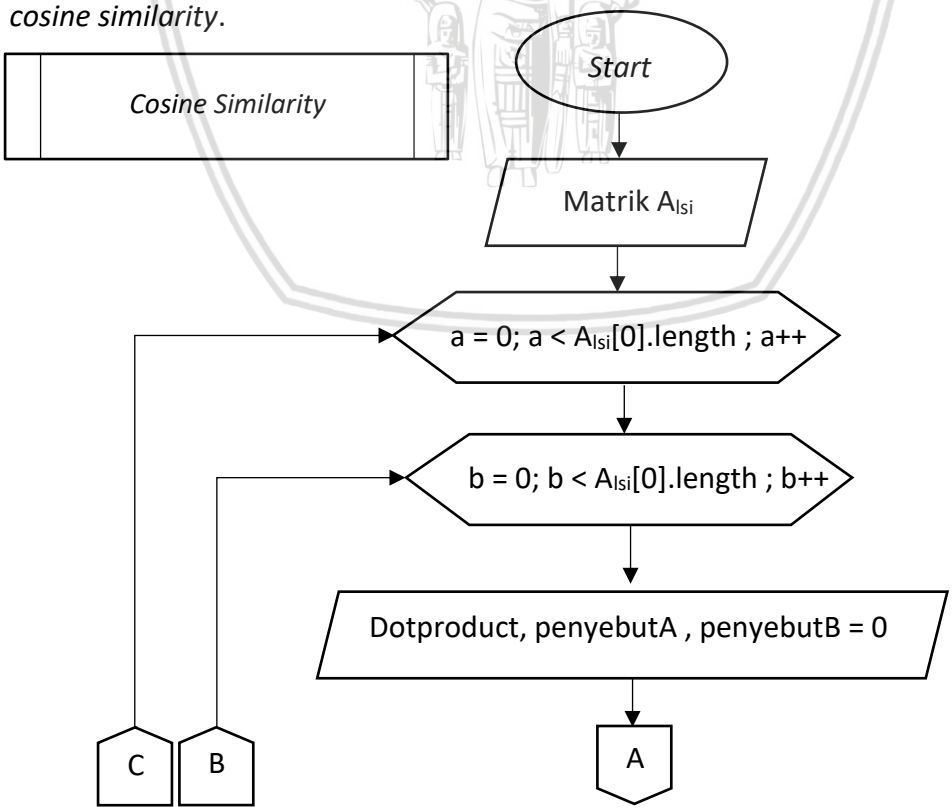


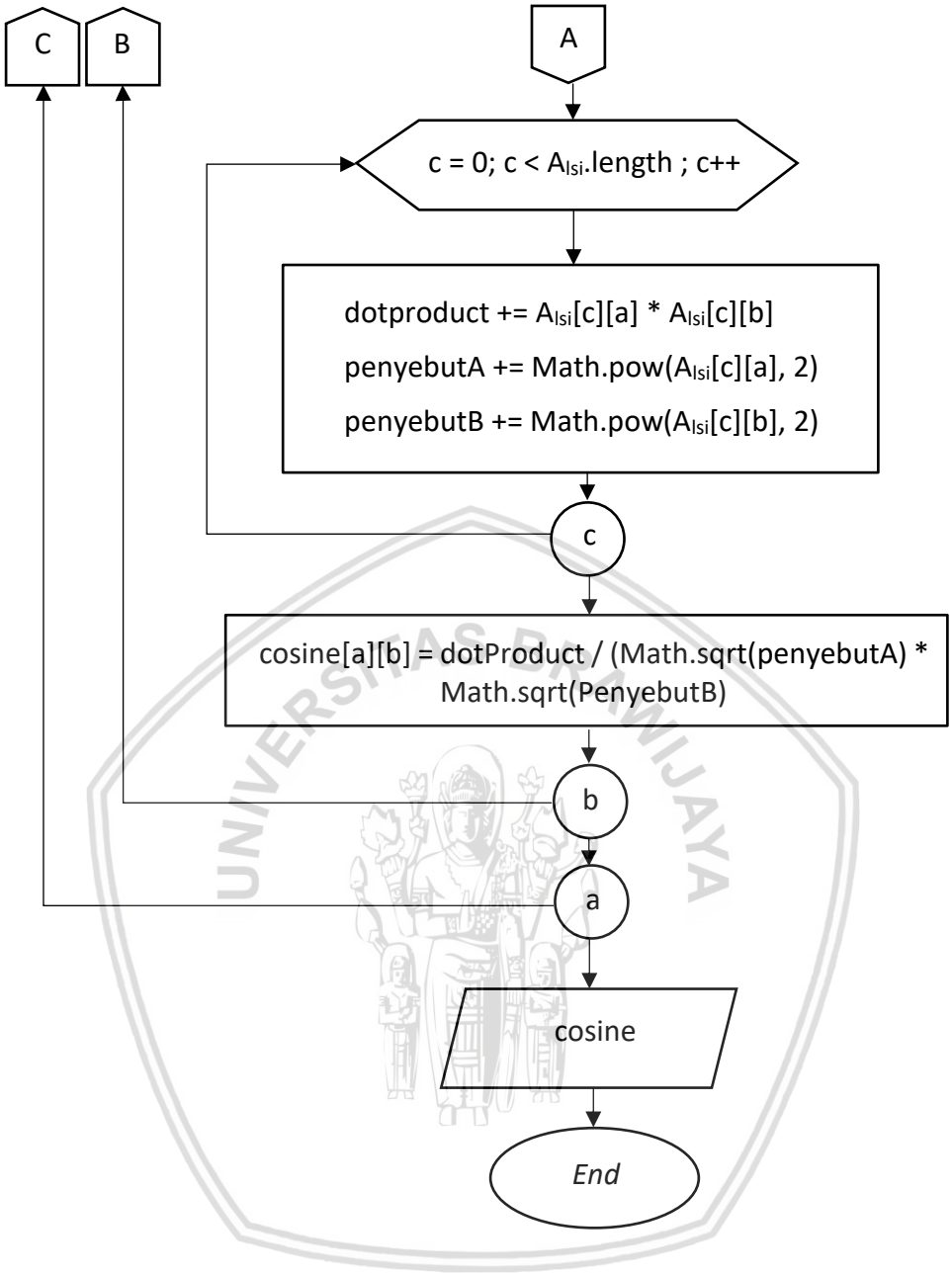


Gambar 4.14 Diagram alir *Latent Semantic Indexing*

4.5 *Vector Space Model* (VSM)

VSM digunakan untuk mengukur kemiripan antar dokumen. VSM mengubah kumpulan dokumen ke dalam bentuk matrik. Pada proses ini rumus *similarity* yang digunakan adalah *cosine similarity*. Gambar 4.15 merupakan diagram alir proses *cosine similarity*.

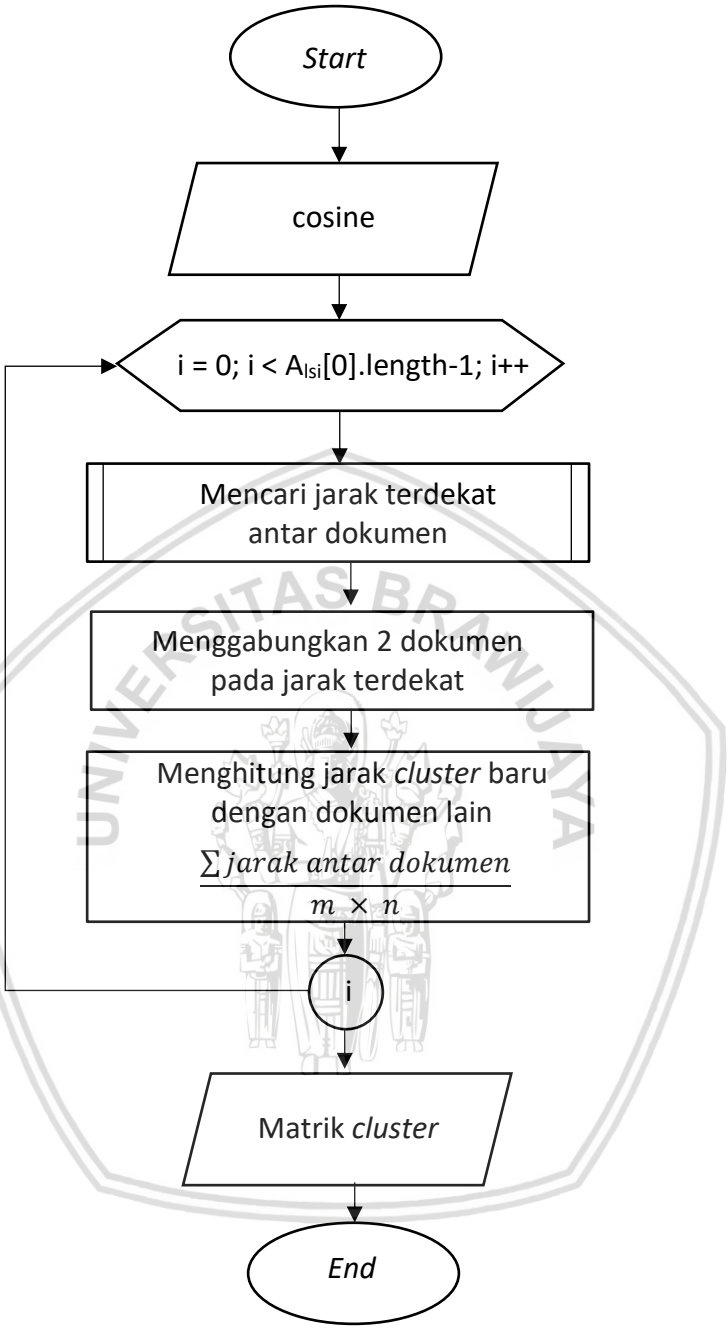




Gambar 4.15 Diagram alir *cosine similarity*

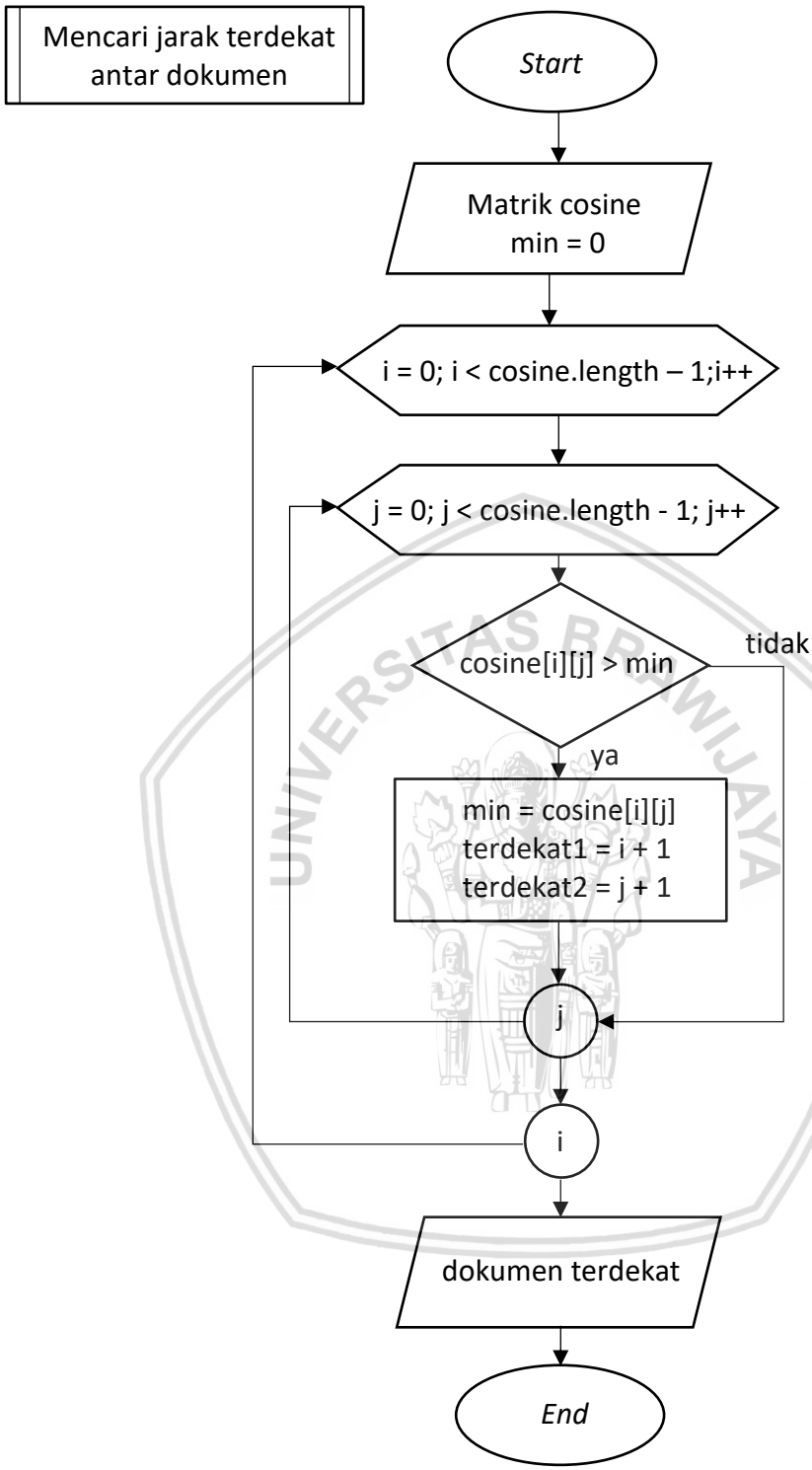
4.6 *Cluster*ing UPGMA

Algoritme *Clustering* UPGMA (*Unweighted Pair Group Method with Arithmetic Mean)* adalah algoritme pengelompokan untuk mengatasi permasalahan pada algoritme *Hierarchical clustering* menggunakan pengukuran rata-rata dalam proses perhitungan jarak antar pasangan data yang telah di*cluster* dengan jarak data yang lainnya. Gambar 4.16 merupakan diagram alir *clustering* UPGMA.



Gambar 4.16 Diagram alir *clustering* UPGMA

Sub proses dari *cluster*ing UPGMA adalah mencari jarak dokumen yang paling terdekat yang artinya dokumen terdekat akan digabung menjadi satu *cluster*, begitu seterusnya hingga semua dokumen tergabung menjadi satu *cluster* untuk semua dokumen. Pada prosesnya, jarak terdekat yang dimaksudkan adalah nilai *cosine similarity* yang terbesar. Gambar 4.17 adalah diagram alir proses perhitungan jarak dokumen yang terdekat.



Gambar 4.17 Diagram alir perhitungan jarak terdekat antar dokumen

4.7 Manualisasi Perhitungan Data

Manualisasi perhitungan data adalah proses penerapan metode reduksi *matrix* menggunakan *Singular Value Decomposition* dan *Latent Semantic Indexing* juga untuk penerapan pengelompokan petisi *online* di situs change.org. Data petisi *online* yang digunakan seperti pada Tabel 4.1.

Tabel 4.1 Data Petisi *Online*

No	Isi petisi *online*
1.	Pak Jokowi, Pasti Bapak Jokowi sudah mendengar, bahwa di Aceh belakangan ini telah menjadi langganan banjir dan tanah longsor
2.	Majelis Ulama Indonesia (MUI) Pusat akhirnya secara resmi telah mengeluarkan Pendapat dan Sikapnya terkait kasus penistaan agama yang dilakukan oleh Basuki Cahaya Purnama alias Ahok di Kepulauan Seribu Jakarta
3.	Pemberian gelar ini diberikan kepada Presiden Jokowi berdasarkan keputusan majelis adat Maluku yang terdiri dari para tetua adat atau Latupati. Presiden menyampaikan rasa terima kasih atas kepercayaan yang diberikan oleh rakyat Maluku
4.	Penetapan Basuki Tjahaya Purnama (Ahok) sebagai tersangka kasus penodaan agama sangat dipengaruhi tekanan dari beberapa orang dan sarat kepentingan politik dibandingkan faktor hukum itu sendiri
5.	Apa yg ingin dilakukan oleh pemerintah dengan menggunakan dana haji untuk kepentingan investasi dan juga infrastruktur sama dengan yg dilakukan penyelenggara ibadah umrah (PIU) / travel nakal
6.	Memblokir Telegram dengan alasan platform itu dijadikan platform komunikasi pendukung terorisme mungkin mirip dengan membakar lumbung padi yang ada tikusnya.

Kemudian selanjutnya akan dilakukan proses *Text Preprocessing* untuk mengubah data yang awalnya belum terstruktur menjadi lebih terstruktur. Proses *Text Preprocessing* terdiri dari *case folding, Tokenizing, Filtering* dan *Stemming*. Tabel 4.2 merupakan hasil implementasi *Text Preprocessing* yaitu *case folding.* Pada proses *case folding* setiap *input* dokumen akan diubah menjadi huruf kecil dan menghapus tanda baca.

Tabel 4.2 Manualisasi proses *case folding*

No	Isi petisi *online*
1.	pak jokowi pasti bapak jokowi sudah mendengar bahwa di aceh belakangan ini telah menjadi langganan banjir dan tanah longsor
2.	majelis ulama indonesia mui pusat akhirnya secara resmi telah mengeluarkan pendapat dan sikapnya terkait kasus penistaan agama yang dilakukan oleh basuki cahaya purnama alias ahok di kepulauan seribu jakarta

Tabel 4.2 Manualisasi proses *case folding* (lanjutan)

No	Isi petisi *online*
3.	pemberian gelar ini diberikan kepada presiden jokowi berdasarkan keputusan majelis adat maluku yang terdiri dari para tetua adat atau latupati presiden menyampaikan rasa terima kasih atas kepercayaan yang diberikan oleh rakyat maluku
4.	penetapan basuki tjahaya purnama ahok sebagai tersangka kasus penodaan agama sangat dipengaruhi tekanan dari beberapa orang dan sarat kepentingan politik dibandingkan faktor hukum itu sendiri
5.	apa yg ingin dilakukan oleh pemerintah dengan menggunakan dana haji untuk kepentingan investasi dan juga infrastruktur sama dengan yg dilakukan penyelenggara ibadah umrah piu travel nakal
6.	memblokir telegram dengan alasan platform itu dijadikan platform komunikasi pendukung terorisme mungkin mirip dengan membakar lumbung padi yang ada tikusnya

Setelah dilakukan proses *case folding* maka langkah selanjutnya merupakan proses *Tokenizing. Tokenizing* digunakan untuk memecah perkata/pertoken berdasarkan spasi supaya dalam pemrosesan selanjutnya akan lebih mudah dan akurat pada pengelompokan dokumen. Tabel 4.3 merupakan manualisasi perhitungan pada proses *Tokenizing*.

Tabel 4.3 Manualisasi proses *Tokenizing*

No	Isi petisi *online*
1.	pak // jokowi // pasti // bapak // jokowi // sudah // mendengar // bahwa // di // aceh // belakangan // ini // telah // menjadi // langganan // banjir // dan // tanah // longsor
2.	majelis // ulama // indonesia // mui // pusat // akhirnya // secara // resmi // telah // mengeluarkan // pendapat // dan // sikapnya // terkait // kasus // penistaan // agama // yang // dilakukan // oleh // basuki // cahaya // purnama // alias // ahok // di // kepulauan // seribu // jakarta
3.	pemberian // gelar // ini // diberikan // kepada // presiden // jokowi // berdasarkan // keputusan // majelis // adat // maluku // yang // terdiri // dari // para // tetua // adat // atau // latupati // presiden // menyampaikan // rasa // terima // kasih // atas // kepercayaan // yang // diberikan // oleh // rakyat // maluku
4.	penetapan // basuki // tjahaya // purnama // ahok // sebagai // tersangka // kasus // penodaan // agama // sangat // dipengaruhi // tekanan // dari // beberapa // orang // dan // sarat // kepentingan // politik // dibandingkan // faktor // hukum // itu // sendiri

Tabel 4.3 Manualisasi proses *Tokenizing* (lanjutan)

No	Isi petisi *online*
5.	apa // yg // ingin // dilakukan // oleh // pemerintah // dengan // menggunakan // dana // haji // untuk // kepentingan // investasi // dan // juga // infrastruktur // sama // dengan // yg // dilakukan // penyelenggara // ibadah // umrah // piu // travel // nakal
6.	memblokir // telegram // dengan // alasan // platform // itu // dijadikan // platform // komunikasi // pendukung // terorisme // mungkin // mirip // dengan // membakar // lumbung // padi // yang // ada // tikusnya

Setelah dilakukan pemecahan dokumen menjadi per kata maka selanjutnya akan dilakukan proses *Filtering. Filtering* merupakan penghapusan kata yang tidak penting tersebut harus didasarkan pada sebuah *stopwords.* Tabel 4.4 merupakan manualisasi proses dari *Filtering*.

Tabel 4.4 Manualisasi proses *Filtering*

No	Isi petisi *online*
1.	jokowi // jokowi // mendengar // aceh // langganan // banjir // tanah // longsor
2.	majelis // ulama // indonesia // mui // pusat // resmi // mengeluarkan // pendapat // sikapnya // terkait // kasus // penistaan // agama // basuki // cahaya // purnama // alias // ahok // kepulauan // seribu // jakarta
3.	pemberian // gelar // presiden // jokowi // berdasarkan // keputusan // majelis // adat // maluku // tetua // adat // latupati // presiden // terima // kasih // kepercayaan // rakyat // maluku
4.	penetapan // basuki // tjahaya // purnama // ahok // tersangka // kasus // penodaan // agama // dipengaruhi // tekanan // orang // sarat // kepentingan // politik // dibandingkan // faktor // hukum
5.	pemerintah // dana // haji // kepentingan // investasi // infrastruktur // penyelenggara // ibadah // umrah // piu // travel // nakal
6.	memblokir // telegram // alasan // platform // dijadikan // platform // komunikasi // pendukung // terorisme // mirip // membakar // lumbung // padi // tikusnya

Setelah menghapus kata-kata yang tidak memiliki makna pada proses *Filtering,* selanjutnya akan dilakukan proses *Stemming.* Pada proses *Stemming,* setiap kata hasil dari *Filtering* akan diubah menjadi bentuk kata dasarnya. Tabel 4.5 merupakan proses manualisasi dari *Stemming*.

Tabel 4.5 Manualisasi proses *Stemming*

No	Isi petisi *online*
1.	jokowi // jokowi // dengar // aceh // langgan // banjir // tanah // longsor
2.	majelis // ulama // indonesia // mui // pusat // resmi // keluar // dapat // sikap // kait // kasus // nista // agama // basuki // cahaya // purnama // alias // ahok // pulau // seribu // jakarta
3.	beri // gelar // presiden // jokowi // dasar // putus // majelis // adat // maluku // tua // adat // latupati // presiden // terima // kasih // percaya // rakyat // maluku
4.	penetapan // basuki // tjahaya // purnama // ahok // sangka // kasus // noda // agama // pengaruh // tekan // orang // sarat // penting // politik // banding // faktor // hukum
5.	pemerintah // dana // haji // penting // investasi // infrastruktur // selenggara // ibadah // umrah // piu // travel // nakal
6.	blokir // telegram // alasan // platform // jadi // platform // komunikasi // dukung // teroris // mirip // bakar // lumbung // padi // tikus

Proses *Stemming* merupakan proses terakhir pada *Text Preprocessing,* di mana *output* terakhirnya merupakan kumpulan token-token yang sudah dalam bentuk kata dasar. Setelah dilakukan proses tersebut tahapan selanjutnya merupakan pembobotan nilai TFIDF. Metode pembobotan ini akan menghitung nilai *dari term frequency* (*TF*) dan juga menghitung nilai dari *Inverse Document Frequency* (*IDF*) pada setiap kata di dalam dokumen. Sebelum terbentuk nilai TF maupun IDF diperlukan menghitung jumlah frekuensi dokumen. Tabel 4.6 merupakan manualisasi pada proses perhitungan frekuensi dokumen.

Tabel 4.6 Manualisasi perhitungan frekuensi dokumen

No	*Term*	Frekuensi Dokumen						DF
		doc 1	doc 2	doc 3	doc 4	doc 5	doc 6	
1.	jokowi	2	0	1	0	0	0	2
2.	dengar	1	0	0	0	0	0	1
3.	aceh	1	0	0	0	0	0	1
4.	langgan	1	0	0	0	0	0	1
5.	banjir	1	0	0	0	0	0	1
6.	tanah	1	0	0	0	0	0	1
7.	longsor	1	0	0	0	0	0	1
8.	majelis	0	1	1	0	0	0	2
9.	ulama	0	1	0	0	0	0	1
10.	indonesia	0	1	0	0	0	0	1
11.	mui	0	1	0	0	0	0	1

Tabel 4.6 Manualisasi perhitungan frekuensi dokumen (lanjutan)

No	*Term*	Frekuensi Dokumen						DF
		doc 1	doc 2	doc 3	doc 4	doc 5	doc 6	
12.	pusat	0	1	0	0	0	0	1
13.	resmi	0	1	0	0	0	0	1
14.	keluar	0	1	0	0	0	0	1
15.	dapat	0	1	0	0	0	0	1
16.	sikap	0	1	0	0	0	0	1
17.	kait	0	1	0	0	0	0	1
18.	kasus	0	1	0	1	0	0	2
19.	nista	0	1	0	0	0	0	1
20.	agama	0	1	0	1	0	0	2
21.	basuki	0	1	0	1	0	0	2
22.	cahaya	0	1	0	0	0	0	1
23.	purnama	0	1	0	1	0	0	2
24.	alias	0	1	0	0	0	0	1
25.	ahok	0	1	0	1	0	0	2
26.	pulau	0	1	0	0	0	0	1
27.	seribu	0	1	0	0	0	0	1
28.	jakarta	0	1	0	0	0	0	1
29.	beri	0	0	1	0	0	0	1
30.	gelar	0	0	1	0	0	0	1
31.	presiden	0	0	2	0	0	0	1
32.	dasar	0	0	1	0	0	0	1
33.	putus	0	0	1	0	0	0	1
34.	adat	0	0	1	0	0	0	1
35.	maluku	0	0	2	0	0	0	1
36.	tua	0	0	1	0	0	0	1
37.	latupati	0	0	1	0	0	0	1
38.	terima	0	0	1	0	0	0	1
39.	kasih	0	0	1	0	0	0	1
40.	percaya	0	0	1	0	0	0	1
41.	rakyat	0	0	1	0	0	0	1
42.	penetapan	0	0	0	1	0	0	1
43.	tjahaya	0	0	0	1	0	0	1
44.	sangka	0	0	0	1	0	0	1
45.	noda	0	0	0	1	0	0	1
46.	pengaruh	0	0	0	1	0	0	1

Tabel 4.6 Manualisasi perhitungan frekuensi dokumen (lanjutan)

No	*Term*	Frekuensi Dokumen						DF
		doc 1	doc 2	doc 3	doc 4	doc 5	doc 6	
47.	tekan	0	0	0	1	0	0	1
48.	orang	0	0	0	1	0	0	1
49.	sarat	0	0	0	1	0	0	1
50.	penting	0	0	0	1	1	0	2
51.	politik	0	0	0	1	0	0	1
52.	banding	0	0	0	1	0	0	1
53.	faktor	0	0	0	1	0	0	1
54.	hukum	0	0	0	1	0	0	1
55.	pemerintah	0	0	0	0	1	0	1
56.	dana	0	0	0	0	1	0	1
57.	haji	0	0	0	0	1	0	1
58.	investasi	0	0	0	0	1	0	1
59.	infrastruktur	0	0	0	0	1	0	1
60.	selenggara	0	0	0	0	1	0	1
61.	ibadah	0	0	0	0	1	0	1
62.	umrah	0	0	0	0	1	0	1
63.	piu	0	0	0	0	1	0	1
64.	travel	0	0	0	0	1	0	1
65.	nakal	0	0	0	0	1	0	1
66.	blokir	0	0	0	0	0	1	1
67.	telegram	0	0	0	0	0	1	1
68.	alasan	0	0	0	0	0	1	1
69.	platform	0	0	0	0	0	2	1
70.	jadi	0	0	0	0	0	1	1
71.	komunikasi	0	0	0	0	0	1	1
72.	dukung	0	0	0	0	0	1	1
73.	teroris	0	0	0	0	0	1	1
74.	mirip	0	0	0	0	0	1	1
75.	bakar	0	0	0	0	0	1	1
76.	lumbung	0	0	0	0	0	1	1
77.	padi	0	0	0	0	0	1	1
78.	tikus	0	0	0	0	0	1	1

Setelah mengetahui jumlah *term* yang terdapat pada masing-masing dokumen, langkah selanjutnya adalah menghitung nilai dari *term frequency* dan *Inverse Document Frequency* (*IDF*) pada setiap kata di dalam dokumen. Tabel 4.7

merupakan manualisasi pada proses perhitungan *term frequency (TF)* , *Inverse Document Frequency (IDF)* dan contoh perhitungan keduanya. Perhitungan tersebut menggunakan rumus Persamaan 2.1 untuk perhitungan TF dan rumus Persamaan 2.2 untuk perhitungan IDF.

Contoh:

- Menghitung nilai *term frequency* (W_{TF}) dengan menggunakan Persamaan 2.1

$$W_{TF} = 1 + log(2)$$
$$= 1 + 0{,}30103$$
$$= 1{,}30103$$

- Menghitung nilai *Inverse Document Frequency* (W_{IDF}) dengan menggunakan Persamaan 2.2

$$W_{IDF} = log\frac{6}{2}$$
$$= log\,3$$
$$= 0{,}477121$$

Tabel 4.7 Manualisasi perhitungan $W_{TF\,(t,d)}$ dan $W_{IDF\,(t,d)}$

No	*Term*	*Term* Frekuensi $W_{TF\,(t,d)}$						$W_{IDF\,(t,d)}$
		doc 1	doc 2	doc 3	doc 4	doc 5	doc 6	
1	jokowi	1,30103	0	1	0	0	0	0,477121
2	dengar	1	0	0	0	0	0	0,778151
3	aceh	1	0	0	0	0	0	0,778151
4	langgan	1	0	0	0	0	0	0,778151
5	banjir	1	0	0	0	0	0	0,778151
6	tanah	1	0	0	0	0	0	0,778151
7	longsor	1	0	0	0	0	0	0,778151
8	majelis	0	1	1	0	0	0	0,477121
9	ulama	0	1	0	0	0	0	0,778151
10	indonesia	0	1	0	0	0	0	0,778151
11	mui	0	1	0	0	0	0	0,778151
12	pusat	0	1	0	0	0	0	0,778151
13	resmi	0	1	0	0	0	0	0,778151
14	keluar	0	1	0	0	0	0	0,778151
15	dapat	0	1	0	0	0	0	0,778151
16	sikap	0	1	0	0	0	0	0,778151
17	kait	0	1	0	0	0	0	0,778151
18	kasus	0	1	0	1	0	0	0,477121
19	nista	0	1	0	0	0	0	0,778151
20	agama	0	1	0	1	0	0	0,477121

Tabel 4.7 Manualisasi perhitungan $W_{TF\ (t,d)}$ dan $W_{IDF\ (t,d)}$ (lanjutan)

No	*Term*	*Term* Frekuensi $W_{TF\ (t,d)}$						$W_{IDF\ (t,d)}$
		doc 1	doc 2	doc 3	doc 4	doc 5	doc 6	
21	basuki	0	1	0	1	0	0	0,477121
22	cahaya	0	1	0	0	0	0	0,778151
23	purnama	0	1	0	1	0	0	0,477121
24	alias	0	1	0	0	0	0	0,778151
25	ahok	0	1	0	1	0	0	0,477121
26	pulau	0	1	0	0	0	0	0,778151
27	seribu	0	1	0	0	0	0	0,778151
28	jakarta	0	1	0	0	0	0	0,778151
29	beri	0	0	1	0	0	0	0,778151
30	gelar	0	0	1	0	0	0	0,778151
31	presiden	0	0	1,30103	0	0	0	0,778151
32	dasar	0	0	1	0	0	0	0,778151
33	putus	0	0	1	0	0	0	0,778151
34	adat	0	0	1	0	0	0	0,778151
35	maluku	0	0	1,30103	0	0	0	0,778151
36	tua	0	0	1	0	0	0	0,778151
37	latupati	0	0	1	0	0	0	0,778151
38	terima	0	0	1	0	0	0	0,778151
39	kasih	0	0	1	0	0	0	0,778151
40	percaya	0	0	1	0	0	0	0,778151
41	rakyat	0	0	1	0	0	0	0,778151
42	penetapan	0	0	0	1	0	0	0,778151
43	tjahaya	0	0	0	1	0	0	0,778151
44	sangka	0	0	0	1	0	0	0,778151
45	noda	0	0	0	1	0	0	0,778151
46	pengaruh	0	0	0	1	0	0	0,778151
47	tekan	0	0	0	1	0	0	0,778151
48	orang	0	0	0	1	0	0	0,778151
49	sarat	0	0	0	1	0	0	0,778151
50	penting	0	0	0	1	1	0	0,477121
51	politik	0	0	0	1	0	0	0,778151
52	banding	0	0	0	1	0	0	0,778151
53	faktor	0	0	0	1	0	0	0,778151
54	hukum	0	0	0	1	0	0	0,778151
55	pemerintah	0	0	0	0	1	0	0,778151
56	dana	0	0	0	0	1	0	0,778151
57	haji	0	0	0	0	1	0	0,778151
58	investasi	0	0	0	0	1	0	0,778151
59	infrastruktur	0	0	0	0	1	0	0,778151

Tabel 4.7 Manualisasi perhitungan $W_{TF\ (t,d)}$ dan $W_{IDF\ (t,d)}$ (lanjutan)

No	*Term*	*Term* Frekuensi $W_{TF\ (t,d)}$						$W_{IDF\ (t,d)}$
		doc 1	doc 2	doc 3	doc 4	doc 5	doc 6	
60	selenggara	0	0	0	0	1	0	0,778151
61	ibadah	0	0	0	0	1	0	0,778151
62	umrah	0	0	0	0	1	0	0,778151
63	piu	0	0	0	0	1	0	0,778151
64	travel	0	0	0	0	1	0	0,778151
65	nakal	0	0	0	0	1	0	0,778151
66	blokir	0	0	0	0	0	1	0,778151
67	telegram	0	0	0	0	0	1	0,778151
68	alasan	0	0	0	0	0	1	0,778151
69	platform	0	0	0	0	0	1,30103	0,778151
70	jadi	0	0	0	0	0	1	0,778151
71	komunikasi	0	0	0	0	0	1	0,778151
72	dukung	0	0	0	0	0	1	0,778151
73	teroris	0	0	0	0	0	1	0,778151
74	mirip	0	0	0	0	0	1	0,778151
75	bakar	0	0	0	0	0	1	0,778151
76	lumbung	0	0	0	0	0	1	0,778151
77	padi	0	0	0	0	0	1	0,778151
78	tikus	0	0	0	0	0	1	0,778151

Langkah selanjutnya adalah menghitung nilai dari pembobotan TFIDF dengan cara mengalikan nilai dari *term frequency (TF)* dengan nilai *Inverse Document Frequency* (*IDF*). Tabel 4.8 merupakan hasil pembobotan dari TFIDF dan contoh perhitungan manualnya.

Contoh:

- Menghitung bobot TFIDF dengan menggunakan Persamaan 2.3

$$tfidf(w) = 1{,}30103 \times 0{,}477$$
$$= 0{,}620749$$

Tabel 4.8 Manualisasi perhitungan pembobotan TFIDF

No	*Term*	Pembobotan TFIDF					
		Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6
1	jokowi	0,620749	0	0,477121	0	0	0
2	dengar	0,778151	0	0	0	0	0
3	aceh	0,778151	0	0	0	0	0

Tabel 4.8 Manualisasi perhitungan pembobotan TFIDF (lanjutan)

No	*Term*	Pembobotan TFIDF					
		Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6
4	langgan	0,778151	0	0	0	0	0
5	banjir	0,778151	0	0	0	0	0
6	tanah	0,778151	0	0	0	0	0
7	longsor	0,778151	0	0	0	0	0
8	majelis	0	0,477121	0,477121	0	0	0
9	ulama	0	0,778151	0	0	0	0
10	indonesia	0	0,778151	0	0	0	0
11	mui	0	0,778151	0	0	0	0
12	pusat	0	0,778151	0	0	0	0
13	resmi	0	0,778151	0	0	0	0
14	keluar	0	0,778151	0	0	0	0
15	dapat	0	0,778151	0	0	0	0
16	sikap	0	0,778151	0	0	0	0
17	kait	0	0,778151	0	0	0	0
18	kasus	0	0,477121	0	0,477121	0	0
19	nista	0	0,778151	0	0	0	0
20	agama	0	0,477121	0	0,477121	0	0
21	basuki	0	0,477121	0	0,477121	0	0
22	cahaya	0	0,778151	0	0	0	0
23	purnama	0	0,477121	0	0,477121	0	0
24	alias	0	0,778151	0	0	0	0
25	ahok	0	0,477121	0	0,477121	0	0
26	pulau	0	0,778151	0	0	0	0
27	seribu	0	0,778151	0	0	0	0
28	jakarta	0	0,778151	0	0	0	0
29	beri	0	0	0,778151	0	0	0
30	gelar	0	0	0,778151	0	0	0
31	presiden	0	0	1,012398	0	0	0
32	dasar	0	0	0,778151	0	0	0
33	putus	0	0	0,778151	0	0	0
34	adat	0	0	0,778151	0	0	0
35	maluku	0	0	1,012398	0	0	0
36	tua	0	0	0,778151	0	0	0
37	latupati	0	0	0,778151	0	0	0
38	terima	0	0	0,778151	0	0	0
39	kasih	0	0	0,778151	0	0	0
40	percaya	0	0	0,778151	0	0	0
41	rakyat	0	0	0,778151	0	0	0
42	penetapan	0	0	0	0,778151	0	0

Tabel 4.8 Manualisasi perhitungan pembobotan TFIDF (lanjutan)

No	*Term*	Pembobotan TFIDF					
		Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6
43	tjahaya	0	0	0	0,778151	0	0
44	sangka	0	0	0	0,778151	0	0
45	noda	0	0	0	0,778151	0	0
46	pengaruh	0	0	0	0,778151	0	0
47	tekan	0	0	0	0,778151	0	0
48	orang	0	0	0	0,778151	0	0
49	sarat	0	0	0	0,778151	0	0
50	penting	0	0	0	0,477121	0,477121	0
51	politik	0	0	0	0,778151	0	0
52	banding	0	0	0	0,778151	0	0
53	faktor	0	0	0	0,778151	0	0
54	hukum	0	0	0	0,778151	0	0
55	pemerintah	0	0	0	0	0,778151	0
56	dana	0	0	0	0	0,778151	0
57	haji	0	0	0	0	0,778151	0
58	investasi	0	0	0	0	0,778151	0
59	infrastruktur	0	0	0	0	0,778151	0
60	selenggara	0	0	0	0	0,778151	0
61	ibadah	0	0	0	0	0,778151	0
62	umrah	0	0	0	0	0,778151	0
63	piu	0	0	0	0	0,778151	0
64	travel	0	0	0	0	0,778151	0
65	nakal	0	0	0	0	0,778151	0
66	blokir	0	0	0	0	0	0,778151
67	telegram	0	0	0	0	0	0,778151
68	alasan	0	0	0	0	0	0,778151
69	platform	0	0	0	0	0	1,012398
70	jadi	0	0	0	0	0	0,778151
71	komunikasi	0	0	0	0	0	0,778151
72	dukung	0	0	0	0	0	0,778151
73	teroris	0	0	0	0	0	0,778151
74	mirip	0	0	0	0	0	0,778151
75	bakar	0	0	0	0	0	0,778151
76	lumbung	0	0	0	0	0	0,778151
77	padi	0	0	0	0	0	0,778151
78	tikus	0	0	0	0	0	0,778151

Setelah pembobotan selesai dilakukan, pada tahap selanjutnya merupakan reduksi matrik dari TFIDF. Reduksi matrik digunakan supaya proses komputasi

lebih cepat dengan mengurangi isi matrik yang digunakan. Pertama matrik TFIDF dipecah menjadi 3 matrik dengan *Singular Value Decomposition* (SVD) berupa matrik *U, S* dan V^T. Kemudian matrik akan diproses dengan menggunakan *Latent Semantic Indexing* (LSI). Pada manualisasi proses SVD dan LSI, hasil manualisasi dihasilkan dari program JAVA dengan menggunakan *library* pemrograman JAMA. Tabel 4.9 merupakan hasil perhitungan dari SVD matrik *U* pada Persamaan 2.4.

Tabel 4.9 Hasil perhitungan *Singular Value Decomposition* matrik *U*

Matrik U					
-0.0168	-0.1673	0.0000	0.0169	-0.0010	0.2961
-0.0011	-0.0146	0.0000	0.0018	-0.0002	0.3884
-0.0011	-0.0146	0.0000	0.0018	-0.0002	0.3884
-0.0011	-0.0146	0.0000	0.0018	-0.0002	0.3884
-0.0011	-0.0146	0.0000	0.0018	-0.0002	0.3884
-0.0011	-0.0146	0.0000	0.0018	-0.0002	0.3884
-0.0011	-0.0146	-0.0000	0.0018	-0.0002	0.3884
-0.1447	-0.1463	0.0000	-0.0577	0.0083	-0.0132
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.1905	0.0300	0.0000	0.0742	-0.0197	0.0004
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.1905	0.0300	0.0000	0.0742	-0.0197	0.0004
-0.1905	0.0300	0.0000	0.0742	-0.0197	0.0004
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.1905	0.0300	0.0000	0.0742	-0.0197	0.0004
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.1905	0.0300	0.0000	0.0742	-0.0197	0.0004

Tabel 4.9 Hasil perhitungan *Singular Value Decomposition* matrik *U* (lanjutan)

Matrik U					
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.2100	0.0152	0.0000	-0.1194	0.0150	0.0008
-0.0259	-0.2539	0.0000	0.0252	-0.0015	-0.0223
-0.0259	-0.2539	0.0000	0.0252	-0.0015	-0.0223
-0.0337	-0.3303	0.0000	0.0328	-0.0019	-0.0290
-0.0259	-0.2539	0.0000	0.0252	-0.0015	-0.0223
-0.0259	-0.2539	0.0000	0.0252	-0.0015	-0.0223
-0.0259	-0.2539	0.0000	0.0252	-0.0015	-0.0223
-0.0337	-0.3303	0.0000	0.0328	-0.0019	-0.0290
-0.0259	-0.2539	0.0000	0.0252	-0.0015	-0.0223
-0.0259	-0.2539	0.0000	0.0252	-0.0015	-0.0223
-0.0259	-0.2539	0.0000	0.0252	-0.0015	-0.0223
-0.0259	-0.2539	0.0000	0.0252	-0.0015	-0.0223
-0.0259	-0.2539	0.0000	0.0252	-0.0015	-0.0223
-0.0259	-0.2539	0.0000	0.0252	-0.0015	-0.0223
-0.1006	0.0337	0.0000	0.2404	-0.0471	-0.0002
-0.1006	0.0337	0.0000	0.2404	-0.0471	-0.0002
-0.1006	0.0337	0.0000	0.2404	-0.0471	-0.0002
-0.1006	0.0337	0.0000	0.2404	-0.0471	-0.0002
-0.1006	0.0337	0.0000	0.2404	-0.0471	-0.0002
-0.1006	0.0337	0.0000	0.2404	-0.0471	-0.0002
-0.1006	0.0337	0.0000	0.2404	-0.0471	-0.0002
-0.1006	0.0337	0.0000	0.2404	-0.0471	-0.0002
-0.0651	0.0227	0.0000	0.1749	0.1509	-0.0001
-0.1006	0.0337	0.0000	0.2404	-0.0471	-0.0002
-0.1006	0.0337	0.0000	0.2404	-0.0471	-0.0002
-0.1006	0.0337	0.0000	0.2404	-0.0471	-0.0002
-0.1006	0.0337	0.0000	0.2404	-0.0471	-0.0002

Tabel 4.9 Hasil perhitungan *Singular Value Decomposition* matrik *U* (lanjutan)

Matrik U					
-0.0055	0.0034	0.0000	0.0448	0.2931	0.0000
-0.0055	0.0034	0.0000	0.0448	0.2931	0.0000
-0.0055	0.0034	0.0000	0.0448	0.2931	0.0000
-0.0055	0.0034	0.0000	0.0448	0.2931	0.0000
-0.0055	0.0034	0.0000	0.0448	0.2931	0.0000
-0.0055	0.0034	0.0000	0.0448	0.2931	0.0000
-0.0055	0.0034	0.0000	0.0448	0.2931	0.0000
-0.0055	0.0034	0.0000	0.0448	0.2931	0.0000
-0.0055	0.0034	0.0000	0.0448	0.2931	0.0000
-0.0055	0.0034	0.0000	0.0448	0.2931	0.0000
-0.0055	0.0034	0.0000	0.0448	0.2931	0.0000
-0.0000	-0.0000	-0.2702	0.0000	-0.0000	0.0000
-0.0000	-0.0000	-0.2702	0.0000	-0.0000	0.0000
-0.0000	-0.0000	-0.2702	0.0000	-0.0000	0.0000
-0.0000	-0.0000	-0.3516	0.0000	-0.0000	0.0000
-0.0000	-0.0000	-0.2702	0.0000	-0.0000	0.0000
-0.0000	-0.0000	-0.2702	0.0000	-0.0000	0.0000
-0.0000	-0.0000	-0.2702	0.0000	-0.0000	0.0000
-0.0000	-0.0000	-0.2702	0.0000	-0.0000	0.0000
-0.0000	-0.0000	-0.2702	0.0000	-0.0000	0.0000
-0.0000	-0.0000	-0.2702	0.0000	-0.0000	0.0000
-0.0000	-0.0000	-0.2702	0.0000	-0.0000	0.0000
-0.0000	-0.0000	-0.2702	0.0000	-0.0000	0.0000

Tabel 4.10 merupakan hasil perhitungan *Singular Value Decomposition* pada matrik S.

Tabel 4.10 Hasil perhitungan *Singular Value Decomposition* matrik *S*

Matrik *S*					
3.3199	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	3.0281	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	2.8794	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	2.8477	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	2.6176	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	2.0004

Tabel 4.11 merupakan hasil perhitungan *Singular Value Decomposition* pada matrik V^T.

Tabel 4.11 Hasil perhitungan *Singular Value Decomposition* matrik V^T

Matrik V^T					
-0.0047	-0.8961	-0.1107	-0.4291	-0.0236	-0.0000
-0.0568	0.0592	-0.9879	0.1310	0.0131	-0.0000
-0.0000	-0.0000	-0.0000	-0.0000	-0.0000	-1.0000
0.0067	-0.4369	0.0922	0.8796	0.1640	-0.0000
-0.0005	0.0504	-0.0049	-0.1583	0.9861	0.0000
0.9984	0.0021	-0.0573	-0.0005	0.0000	0.0000

Setelah matrik pada TFIDF dipecah menjadi 3 matrik. Langkah selanjutnya adalah memotong matrik hasil *Singular Value Decomposition.* Matrik yang dipotong hanya pada matrik *S* dan V^T karena pada kedua matrik tersebut akan digunakan pada rumus Persamaan 2.5 pada proses LSI. Tabel 4.12 merupakan hasil pemotongan matrik pada matrik *S* dan Tabel 4.13 merupakan hasil pemotongan pada matrik V^T .

Tabel 4.12 Hasil pemotongan matrik pada matrik *S*

Matrik *S*			
3.3199	0.0000	0.0000	0.0000
0.0000	3.0281	0.0000	0.0000
0.0000	0.0000	2.8794	0.0000
0.0000	0.0000	0.0000	2.8477

Tabel 4.13 Hasil pemotongan matrik pada matrik V^T

Matrik V^T					
-0.0047	-0.8961	-0.1107	-0.4291	-0.0236	-0.0000
-0.0568	0.0592	-0.9879	0.1310	0.0131	-0.0000
-0.0000	-0.0000	-0.0000	-0.0000	-0.0000	-1.0000
0.0067	-0.4369	0.0922	0.8796	0.1640	-0.0000

Pada Tabel 4.12 dan 4.13 dicontohkan untuk pemotongan matrik *S* dilakukan pemotongan sebanyak 2 baris dan 2 kolom. Sedangkan pada matrik *V transpose* pemotongan dilakukan pada baris sebanyak 2. Langkah selanjutnya adalah proses *Latent Semantic Indexing* dengan perhitungan menggunakan Persamaan 2.5.

$A_{lsi} =$

$$= \begin{bmatrix} -0.0047 & -0.0568 & -0.00 & 0.0067 \\ -0.8961 & 0.0592 & -0.00 & -0.4369 \\ -0.1107 & -0.9879 & -0.00 & 0.0922 \\ -0.4291 & 0.1310 & -0.00 & 0.8796 \\ -0.0236 & 0.0131 & -0.00 & 0.1640 \\ -0.0000 & -0.0000 & -1.00 & -0.0000 \end{bmatrix} \begin{bmatrix} -0.0047 & -0.0568 & -0.00 & 0.0067 \\ -0.8961 & 0.0592 & -0.00 & -0.4369 \\ -0.1107 & -0.9879 & -0.00 & 0.0922 \\ -0.4291 & 0.1310 & -0.00 & 0.8796 \end{bmatrix}$$

$$= \begin{bmatrix} -0.0155 & -0.1720 & 0.0000 & 0.0190 \\ -2.9751 & 0.1793 & 0.0000 & -1.2442 \\ -0.3674 & -2.9914 & 0.0000 & 0.2627 \\ -1.4245 & 0.3967 & 0.0000 & 2.5047 \\ -0.0785 & 0.0396 & 0.0000 & 0.4671 \\ 0.0000 & 0.0000 & -2.8794 & 0.0000 \end{bmatrix}$$

Pada matrik A_{lsi} jumlah baris menandakan jumlah dokumen yang diproses. Tabel 4.14 merupakan hasil pada proses *Latent Semantic Indexing*.

Tabel 4.14 Hasil perhitungan reduksi matrik

Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6
-0.0155	-2.9751	-0.3674	-1.4245	-0.0785	0.0000
-0.1720	0.1793	-2.9914	0.3967	0.0396	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	-2.8794
0.0190	-1.2442	0.2627	2.5047	0.4671	0.0000

Proses selanjutnya adalah menghitung *cosine similarity* antar dokumen. Tabel 4.15 merupakan hasil perhitungan dari *cosine similarity* antar dokumen dengan Persamaan 2.6 dan contoh perhitungan manual.

Contoh:

- Menghitung *cosine similarity* dengan menggunakan Persamaan 2.6

$$cos(doc1, doc2) =$$

$$= \frac{-0{,}0155 * -2{,}9751 + -0{,}1720 * 0{,}1793 + 0{,}0 * 0{,}0 + 0{,}0190 * -1{,}2442}{\sqrt{-0{,}0155^2 + -0{,}1720^2 + 0{,}0^2 + 0{,}0190^2} * \sqrt{-0{,}0155^2 + -0{,}1720^2 + 0{,}0^2 + 0{,}0190^2}}$$

$$= \frac{0{,}046114 + (-0{,}03084) + 0 + (-0{,}02364)}{\sqrt{0{,}00024 + 0{,}029584 + 0 + 0{,}000361} * \sqrt{0{,}00024 + 0{,}029584 + 0 + 0{,}000361}}$$

$$= \frac{-0{,}00837}{\sqrt{0{,}030185} * \sqrt{10{,}4314}}$$

$$= \frac{-0{,}00837}{0{,}173739 * 3{,}229768}$$

$$= \frac{-0{,}00837}{0{,}561137} = -0{,}01491$$

Tabel 4.15 Hasil perhitungan *cosine similarity*

cosine similarity						
cosim	doc 1	doc 2	doc 3	doc 4	doc 5	doc 6
doc 1	1	-0,01491	0,999226	0,125524	0,039725	0
doc 2	-0,01491	1	0,023523	0,121281	-0,22182	0
doc 3	0,999226	0,023523	1	0,121894	0,023011	0
doc 4	0,125524	0,121281	0,121894	1	0,936957	0
doc 5	0,039725	-0,22182	0,023011	0,936957	1	0
doc 6	0	0	0	0	0	1

Langkah selanjutnya akan dilakukan proses *clustering* dengan metode UPGMA, pada metode ini akan dilakukan perulangan sebanyak *n*-1 proses. Dokumen akan digabungkan sesuai jarak dokumen terdekat hingga membentuk satu *cluster*. Tabel 4.16 merupakan hasil dari penggabungan *cluster* terdekat pada iterasi pertama dan contoh menghitung jarak antara *cluster* yang baru dengan dokumen yang lain.

Contoh:

- Menghitung jarak *cluster* baru dokumen 1(3) dengan dokumen 2 dengan menggunakan Persamaan 2.7.

$$d_{UPGMA} doc\ (1,3),\ doc\ 2 =$$

$$= \frac{(-0{,}01491)+(0{,}023523)}{2 \times 1}$$

$$= \frac{0{,}008615}{2}$$

$$= 0{,}004307$$

Tabel 4.16 Hasil pembentukan *cluster* iterasi 1

cosim	doc (1,3)	doc 2	doc 4	doc 5	doc 6
doc (1,3)	1	0,004307	0,123709	0,031368	0
doc 2	0,004307	1	0,121281	-0,22182	0
doc 4	0,123709	0,121281	1	0,936957	0
doc 5	0,031368	-0,22182	0,936957	1	0
doc 6	0	0	0	0	1

Tabel 4.17 merupakan hasil dari penggabungan *cluster* terdekat pada iterasi kedua.

Tabel 4.17 Hasil pembentukan *cluster* iterasi 2

cosim	doc (4,5)	doc (1,3)	doc 2	doc 6
doc (4,5)	1	0,077539	-0,05027	0
doc (1,3)	0,077539	1	0,004307	0
doc 2	-0,05027	0,004307	1	0
doc 6	0	0	0	1

Tabel 4.18 merupakan hasil dari penggabungan *cluster* terdekat pada iterasi ketiga.

Tabel 4.18 Hasil pembentukan *cluster* iterasi 3

cosim	doc ((4,5)(1,3))	doc 2	doc 6
doc ((4,5)(1,3))	1	-0,02298	0
doc 2	-0,02298	1	0
doc 6	0	0	1

Tabel 4.19 merupakan hasil dari penggabungan *cluster* terdekat pada iterasi keempat.

Tabel 4.19 Hasil pembentukan *cluster* iterasi 4

cosim	doc (((4,5)(1,3)),6)	doc 2
doc (((4,5)(1,3)),6)	1	-0,01149
doc 2	-0,011490543	1

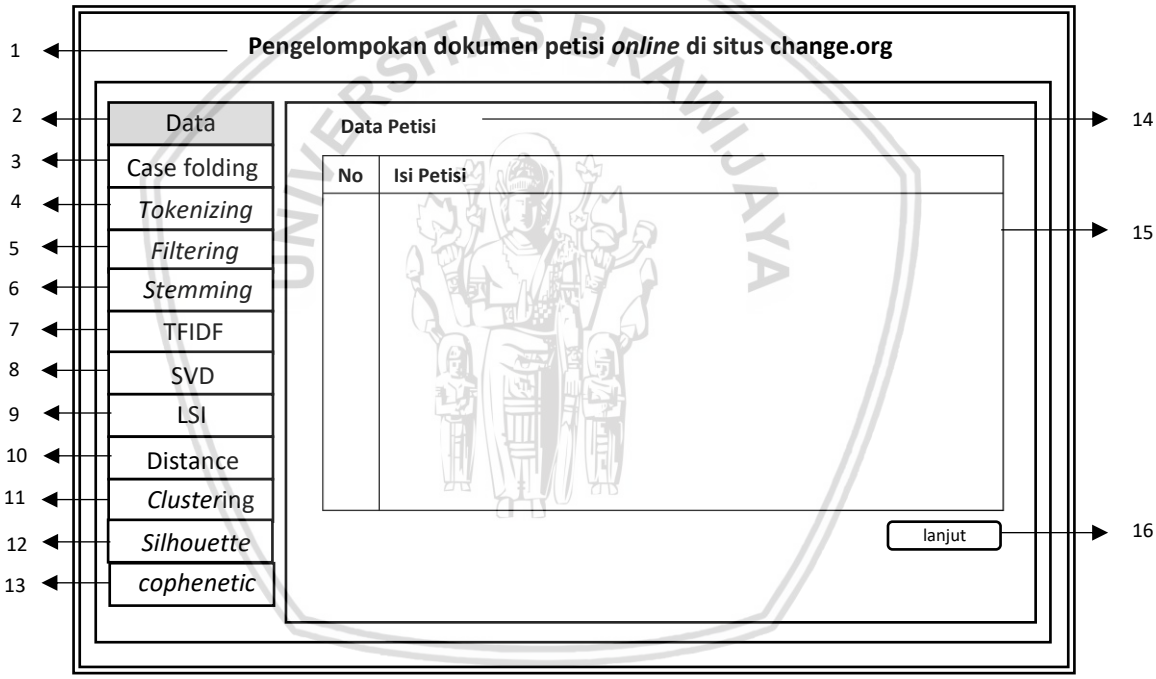
Pada iterasi kelima atau yang terakhir akan menggabungkan dokumen hasil penggabungan dokumen 4, 5, 1, 3 dan 6 dengan dokumen 2. Tabel 4.20 adalah proses penggabungan terakhir pada pembentukan *cluster* dengan metode UPGMA *cluster*ing.

Tabel 4.20 Hasil pembentukan *cluster* iterasi 5

cosim	doc ((((4,5)(1,3)),6),2)
doc ((((4,5)(1,3)),6),2)	1

4.8 Perancangan Antarmuka Sistem

Pada *sub*bab ini menjelaskan mengenai perancangan antarmuka sistem pengelompokan dokumen petisi *online* di situs change.org menggunakan algoritme *hierarchical clustering* UPGMA. Gambar 4.18 merupakan rancangan antarmuka sistem.



Gambar 4.18 Perancangan antarmuka sistem

Komponen yang terdapat pada gambar 4.18 akan dijelaskan berdasarkan label angka tersebut.

1. Menampilkan Judul penelitian.
2. Menampilkan *input* data petisi dan menampilkan isi dari petisi.
3. Menampilkan hasil dari proses *case folding.*
4. Menampilkan hasil dari proses *Tokenizing.*
5. Menampilkan hasil dari proses *Filtering.*
6. Menampilkan hasil dari proses *Stemming.*
7. Menampilkan hasil dari proses pembobotan TFIDF.
8. Menampilkan hasil dari proses *Singular Value Decomposition.*

9. Menampilkan hasil dari proses *Latent Semantic Indexing.*
10. Menampilkan hasil dari proses *cosine similarity.*
11. Menampilkan hasil dari proses pengelompokan dengan metode UPGMA *clustering.*
12. Menampilkan hasil dari proses penentuan evaluasi nilai *Silhouette Coefficient.*
13. Menampilkan hasil dari proses penentuan evaluasi nilai *Cophenetic Correlation Coefficient*.
14. Menampilkan *sub* judul .
15. Menampilkan isi dari *sub* menu berupa tabel.
16. Menampilkan tombol untuk melakukan proses selanjutnya.

4.9 Perancangan Pengujian

Pada tahapan ini akan dilakukan pengujian dari hasil *clustering* UPGMA pada sistem untuk dianalisis keakuratannya terhadap hasil *clustering*. Pada perancangan pengujian terdapat 2 pengujian yang dilakukan. Pertama dengan menggunakan metode evaluasi *Cophenetic Correlation Coefficient* dan yang kedua dengan menggunakan metode *Silhouette Coefficient.*

4.9.1 Rancangan Pengujian *Cophenetic Correlation Coefficient*

Pengujian yang pertama adalah dengan menggunakan algoritme *Cophenetic Correlation Coefficient*. Algoritme ini diperlukan untuk melihat kualitas (*quality*) hasil analisis pada *cluster*. *Cophenetic Correlation Coefficient* merupakan koefisien korelasi antara matrik jarak dan matrik jarak *dendrogram* (matriks *cophenetic*). Skenario *rank* matrik LSI yang digunakan mulai dari 100% hingga 5%. Rancangan pengujian ini juga akan membandingkan nilai nilai *cophenetic* antara hasil reduksi matrik dengan matrik tanpa proses reduksi matrik. Tabel 4.21 merupakan rancangan pengujian *Cophenetic Correlation Coefficient.*

Tabel 4.21 Rancangan pengujian *Cophenetic Correlation Coefficient*

No	*Rank*	Nilai *Cophenetic*
1	Non LSI	
2	100%	
3	95%	
4	90%	
5	85%	
6	80%	
7	75%	
8	70%	
9	65%	
10	60%	
11	55%	
12	50%	
13	45%	

14	40%	
15	35%	
16	30%	
17	25%	
18	20%	
19	15%	
20	10%	
21	5%	

4.9.2 Rancangan Pengujian *Silhouette Coefficient*

Pengujian kedua yang dilakukan adalah dengan menggunakan Algoritme evaluasi *Silhouette Coefficient* pada Persamaan 2.11. Algoritme tersebut digunakan untuk mengetahui jumlah *cluster* yang memiliki tingkat validitas tertinggi sehingga paling sesuai untuk digunakan. Pada pengujian ini, matrik jarak yang digunakan adalah matrik jarak yang memiliki nilai *cophenetic* tertinggi pada pengujian pertama. Nilai yang dijadikan acuan dari *Silhouette Coefficient* adalah nilai yang terbesar dan mendekati nilai 1. Rancangan pengujian dengan *Silhouette Coefficient* terdiri dari skenario *cluster* mulai dari 100 *cluster* hingga 2 *cluster*. Tabel 4.22 merupakan rancangan pengujian *Silhouette Coefficient*.

Tabel 4.22 Rancangan pengujian *Silhouette Coefficient*

No	Jumlah *Cluster*	Nilai *Silhouette*
1	100	
2	99	
3	98	
4	97	
5	96	
6	95	
7	94	
8	93	
9	92	
10	91	
…	…	…
97	4	
98	3	
99	2	

4.10 Penarikan Kesimpulan

Proses penarikan kesimpulan akan dilakukan saat semua tahapan proses perancangan, implementasi dan pengujian telah selesai dilakukan. Penarikan kesimpulan dihasilkan dari analisis pada sistem yang sudah dibuat. Setelah penarikan kesimpulan dilakukan, maka dapat dibuat saran terhadap sistem guna perbaikan pada sistem yang dapat dilakukan pada pengembangan selanjutnya.

BAB 5 IMPLEMENTASI

Pada bab implementasi ini, akan dibahas mengenai hasil implementasi sistem pengelompokan dokumen petisi *online* di situs change.org menggunakan algoritme *hierarchical clustering* UPGMA yang berdasarkan kebutuhan analisis dan perancangan pada bab sebelumnya. *Sub* bab yang akan dibahas antara lain spesifikasi sistem, implementasi perangkat program serta implementasi antarmuka.

5.1 Spesifikasi sistem

Sistem merupakan suatu gambaran dari perpaduan kebutuhan dari perangkat lunak dan kebutuhan perangkat keras. Kebutuhan tersebut dijadikan acuan untuk melakukan implementasi pada sistem. Adapun spesifikasi sistem diimplementasikan pada spesifikasi perangkat lunak *(software)* dan spesifikasi perangkat keras *(hardware)*

5.1.1 Spesifikasi Perangkat Lunak

Pembangunan sistem pengelompokan dokumen petisi *online* di situs change.org menggunakan algoritme *hierarchical cluster*ing UPGMA didukung oleh perangkat lunak dengan spesifikasi setiap komponennya pada Tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat Lunak

Nama	Spesifikasi
Sistem Operasi	Microsoft Windows 10 Pro 64-bit
Bahasa Pemrograman	JAVA
Aplikasi Pemrograman	Netbeans IDE 8.01

5.1.2 Spesifikasi Perangkat Keras

Pembangunan sistem pengelompokan dokumen petisi *online* di situs change.org menggunakan algoritme *hierarchical cluster*ing UPGMA didukung oleh perangkat keras dengan spesifikasi setiap komponennya pada Tabel 5.2.

Tabel 5.2 Spesifikasi Perangkat Keras

Nama Komponen	Spesifikasi
Processor	Intel® Core™ i5-5200U CPU @ 2.20GHz 2.19 GHz
RAM	4,00 GB DDR3
Hardisk	500 GB

5.2 Implementasi Program

Pada implementasi program menjelaskan tahapan proses dari sistem pengelompokan petisi *online* disitus change.org menggunakan algortima hierarchical *clustering* UPGMA yang dijalankan berdasarkan perancangan pada bab sebelumnya. Proses utama pada sistem ini adalah *Text Preprocessing*, pembobotan TFIDF, *Singular Value Decomposition , Latent Semantic Indexing, cosine similarity, clustering* UPGMA, *Silhouette Coefficient* dan *Cophenetic Correlation Coefficient*. Penggunaan setiap proses tersebut direpresentasikan pada setiap modul pemrograman atau fungsi. Tabel 5.3 merupakan daftar fungsi implementasi program yang digunakan.

Tabel 5.3 Daftar Fungsi Implementasi Program

No	Proses	Nama Fungsi	Keterangan
1.	*Text Preprocessing*	`lowercase()`	Berfungsi untuk mengubah huruf besar menjadi huruf kecil
		`hapusnonabjad()`	Berfungsi untuk menghapus karakter selain abjad
		`Tokenizing()`	Berfungsi untuk memecah kalimat *input* menjadi susunan kata atau token berdasarkan spasi
		`Filtering()`	Berfungsi untuk menghapus kata-kata yang tidak memiliki makna atau kata yang sama dengan kata pada variable array stopword dihapuskan
		`Stemmingnazief()`	Berfungsi untuk mengubah kata yang memiliki imbuhan menjadi kata dasar
2.	Pembobotan TFIDF	`pembentukantermunik()`	Berfungsi untuk membentuk *term* unik pada seluruh dokumen *input*

		`rawTF()`	Berfungsi untuk menghitung jumlah *term* yang muncul pada setiap dokumen
		`pembobotanTF()`	Berfungsi untuk menghitung pembobotan *term frequency* pada setiap dokumen
		`perhitunganDF()`	Berfungsi untuk menghitung nilai dari DF atau dokumen frekuensi yang mengandung suatu fitur kata
		`pembobotanIDF()`	Berfungsi untuk menghitung pembobotan IDF
		`pembobotanTFIDF()`	Berfungsi untuk menghitung pembobotan TFIDF, mengkalikan nilai TF dengan nilai IDF
3.	*Singular Value Decomposition*	`SVD()`	Berfungsi untuk memecah matrik TFIDF menjadi matrik U, matrik S dan matrik V *transpose*
4.	*Latent Semantic Indexing*	`reducematrixLSI()`	Berfungsi untuk mereduksi matrik dari *Singular Value Decomposition* dan membentuk matrik LSI dari perkalian matrik V dengan matrik S
5.	*Cosine similarity*	`hitungcosinesimilarity()`	Berfungsi untuk menghitung nilai *similarity* antar dokumen
6.		`caridekat()`	Berfungsi untuk menghitung jarak

	Clustering UPGMA		antar dokumen yang terdekat berdasarkan nilai *cosine similarity* terbesar
		`update()`	Berfungsi untuk menghapus kedua dokumen yang memiliki nilai *cosine similarity* terbesar dan menambahkan tempat pada index awal penyimpanan *lingkedlist* untuk gabungan kedua dokumen sebelumnya yang sudah dihapus atau *cluster* baru
		`updatearray()`	Berfungsi untuk mengupdate nilai pada array penyimpanan nilai *cosine similarity* antar dokumen setelah proses penghapusan 2 dokumen yang terdekat
		`jarak()`	Berfungsi untuk penjumlahan jarak dokumen *cluster* baru dengan dokumen lainnya
		`ratarata()`	Berfungsi untuk menghitung jarak rata-rata *cluster* baru dengan dokumen lainnya
7.	*Cophenetic Correlation Coefficient*	`Copheneticcorrelation()`	Berfungsi untuk menghitung nilai *cophenetic* pada hasil *cluster*ing
8.		`evaluasiSilhouette()`	Berfungsi untuk menghitung dan

	Silhouette Coefficient		mengukur kualitas suatu *cluster*.
		Hitungai()	Berfungsi untuk menghitung nilai rata-rata jarak antara *object* I dengan *object* lainnya dalam satu *cluster* yang sama.
		Hitungbi()	Berfungsi untuk menghitung nilai minimum rata-rata jarak antara *object* I dengan *object* lainnya dalam *cluster* yang berbeda.

5.3 *Text Preprocessing*

Pada proses *Text Preprocessing* ada 5 tahapan yang harus dilalui, yaitu *lowercase*, penghapusan non abjad, *Tokenizing*, *Filtering*, dan *Stemming.*

5.3.1 Fungsi *lowercase*

Fungsi ***lowercase*** adalah proses untuk mengubah semua huruf besar menjadi huruf kecil dan hasilnya di *input* pada tabel. Potongan program fungsi ***lowercase*** ditunjukan pada Kode Program 5.1 berikut.

```
Public void lowercase() {
        DefaultTabelModel tabelModel = new
DefaultTabelModel(header1, data.length);
        jTabel3.setModel(tabelModel);
        for (int i = 0; i < data.length; i++) {
            jTabel3.setRowHeight(i, 30);
            jTabel3.setValueAt(i + 1, I, 0);
            jTabel3.setValueAt(data[i].toLowercase(), i, 1);
        }
        int simpansementara = 0;
        String[] temp;
        for (int i = 0; i < data.length; i++) {
            temp = String.valueOf(jTabel3.getValueAt(i,
1)).split("");
            if (temp.length > simpansementara) {
                simpansementara = temp.length;
            } else {
                temp = null;
            }
        }
        jTabel3.setAutoResizeMode(0);
        TabelColumnModel columnModel = jTabel3.getColumnModel();
        columnModel.getColumn(0).setPreferredWidth(2 * 15);
```

```
        columnModel.getColumn(1).setPreferredWidth(6 *
simpansementara);
    }
```

Kode Program 5.1 *lowercase*

5.3.2 Fungsi hapusnonabjad

Fungsi **hapusnonabjad** adalah proses untuk menghapus karakter selain abjad. Karakter yang terdeteksi sebagai karakter non abjad tersebut akan digantikan dengan karakter spasi. Potongan program fungsi **hapusnonabjad** ditunjukkan pada Kode Program 5.2.

```
  public void hapusnonabjad() {
        DefaultTabelModel        tabelModel2        =        new
DefaultTabelModel(header1, data.length);
        jTabel2.setModel(tabelModel2);
        jTabel2.setAutoResizeMode(0);
        TabelColumnModel columnModel = jTabel2.getColumnModel();
        columnModel.getColumn(0).setPreferredWidth(40);
        columnModel.getColumn(1).setPreferredWidth(6 * 9513);

        for (int i = 0; i < data.length; i++) {
            jTabel2.setRowHeight(i, 30);
            jTabel2.setValueAt(i + 1, i, 0);
            String hasil = String.valueOf(jTabel3.getValueAt(i,
1)).replaceAll("[^a-zA-Z]", " ");
            jTabel2.setValueAt(hasil, i, 1);
        }
    }
```

Kode Program 5.2 hapusnonabjad

5.3.3 Fungsi *Tokenizing*

Fungsi ***Tokenizing*** adalah proses untuk memecah kata pada kalimat yang telah di *input*. Kata dipecah berdasarkan karakter spasi pada kalimat *input*. Kata hasil *Tokenizing* di *input* pada tabel *Tokenizing*. Potongan program fungsi ***Tokenizing*** ditunjukan seperti Kode Program 5.3.

```
public void Tokenizing() {
        DefaultTabelModel        tabelModel4        =        new
DefaultTabelModel(header1, data.length);
        jTabel4.setModel(tabelModel4);
        jTabel4.setAutoResizeMode(0);
        TabelColumnModel columnModel = jTabel4.getColumnModel();
        columnModel.getColumn(0).setPreferredWidth(40);
        columnModel.getColumn(1).setPreferredWidth(6 * 9513);
        String append = "";String[] hasiltoken;
                for (int i = 0; i < data.length; i++) {
            jTabel4.setRowHeight(i, 30);
            jTabel4.setValueAt(i + 1, I, 0);
            hasiltoken   =   String.valueOf(jTabel2.getValueAt(i,
1)).trim().split("\\s+");
            for (int j = 0; j < hasiltoken.length; j++) {
                if (j == (hasiltoken.length - 1)) {
                    append = append + (hasiltoken[j]);
```

```
                } else {
                    append = append + (hasiltoken[j] + " // ");
                }}
                        jTabel4.setValueAt(append, i, 1);
            append = "";
        }
    }
```

Kode Program 5.3 *Tokenizing*

5.3.4 Fungsi *Filtering*

Fungsi ***Filtering*** adalah proses penghapusan kata yang tidak memiliki makna. Kata yang tidak memiliki makna tersebut adalah kumpulan kata yang terdapat pada array *stopWords*. Jika kata pada proses *Tokenizing* sama dengan salah satu kata pada *stopWords* maka kata tersebut dihapuskan. Hasil dari proses *Filtering* akan di *input* pada tabel *Filtering*. Potongan program fungsi ***Filtering*** ditunjukan pada Kode Program 5.4.

```
    public void Filtering() {
        DefaultTabelModel tabelModel5 = new
DefaultTabelModel(header1, data.length);
        jTabel5.setModel(tabelModel5);
        jTabel5.setAutoResizeMode(0);
        TabelColumnModel columnModel = jTabel5.getColumnModel();
        columnModel.getColumn(0).setPreferredWidth(40);
        columnModel.getColumn(1).setPreferredWidth(6 * 9513);
        String append = "";  String[] hasilfilter;
              for (int i = 0; i < data.length; i++) {
            jTabel5.setRowHeight(i, 30);
            jTabel5.setValueAt(i + 1, i, 0);
            hasilfilter  =  String.valueOf(jTabel4.getValueAt(i,
1)).trim().split(" // ");
            for (int j = 0; j < hasilfilter.length; j++) {
                for (int k = 0; k < stopWords.length; k++) {
                    if  (!hasilfilter[j].equals(stopWords[k])  &&
k == (stopWords.length - 1)) {
                        if (j == (hasilfilter.length - 1)) {
                            append = append + (hasilfilter[j]);
                            break;
                        } else {
                            append = append + (hasilfilter[j] +
" // ");
                            break; }
                                            }else               if
(hasilfilter[j].equals(stopWords[k])) {
                        break;
                    }   }  }
                        jTabel5.setValueAt(append, i, 1);
            append = "";
        }
    }
```

Kode Program 5.4 *Filtering*

5.3.5 Fungsi *Stemming*nazief

Fungsi ***Stemming*nazief** adalah proses transformasi kata yang memiliki imbuhan menjadi kata dasar. Hasil dari proses *Stemming* dengan menggunakan

metode *Stemming* Nazief Adriani kemudian akan di *input* pada tabel. Potongan program fungsi ***Stemming*nazief** ditunjukan pada Kode Program 5.5.

```
public void Stemmingnazief() {
        StemmingNaziefAndriani stem = null;
        try {
            stem = new StemmingNaziefAndriani();
        }        catch        (IOException        ex)        {
Logger.getLogger(skripsiform.class.getName()).log(Level.SEVERE,
null, ex);
        }

        jTabbedPane1.setSelectedIndex(4);
        DefaultTabelModel        tabelModel6        =        new
DefaultTabelModel(header1, data.length);
        jTabel6.setModel(tabelModel6);
        jTabel6.setAutoResizeMode(0);
        TabelColumnModel columnModel = jTabel6.getColumnModel();
        columnModel.getColumn(0).setPreferredWidth(40);
        columnModel.getColumn(1).setPreferredWidth(6 * 9513);
        String append = "";
        String[] hasiltoken;

        for (int i = 0; i < data.length; i++) {
            jTabel6.setRowHeight(i, 30);
            jTabel6.setValueAt(i      +      1,      i,      0);
jTabel5.setValueAt(String.valueOf(jTabel5.getValueAt(i,
1)).replaceAll("      |      |      |      |      |      |
", " "), i, 1);

            hasiltoken   =   String.valueOf(jTabel5.getValueAt(i,
1)).trim().split(" // ");
            for (int j = 0; j < hasiltoken.length; j++) {
                if (j == (hasiltoken.length - 1)) {
                    append          =          append          +
stem.KataDasar(hasiltoken[j]);
                } else {
                    append          =          append          +
(stem.KataDasar(hasiltoken[j]) + " // ");
                }
            }

            jTabel6.setValueAt(append, i, 1);
            append = "";
        }
    }
```

Kode Program 5.5 Stemmingnazief

5.4 Pembobotan TFIDF

Setelah fungsi-fungsi pada *Text Preprocessing* sudah diproses maka langkah setelahnya merupakan pembobotan TFIDF. Pembobotan ini bertujuan untuk mengubah data yang awalnya berupa *text* menjadi data *numerik.* Beberapa proses yang berkaitan dengan pembobotan TFIDF antara lain dengan menggunakan fungsi **pembentukan*term*unik**, fungsi **rawTF**, fungsi **pembobotanTF**, fungsi **perhitunganDF**, fungsi **pembobotanIDF** dan fungsi **pembobotanTFIDF**.

5.4.1 Fungsi pembentukan*term*unik

Fungsi **pembentukan*term*unik** adalah proses untuk membentuk kata-kata yang unik pada semua dokumen *input*. Kata unik atau *term* tersebut disimpan pada variabel penyimpanan *linkedlist* dengan nama *hasilterm*. Pembentukan *term* unik memudahkan pada proses selanjutnya di mana setiap dokumen akan dihitung berdasarkan banyaknya *term*. Potongan program fungsi **pembentukan*term*unik** ditunjukan pada Kode Program 5.6.

```
public void pembentukantermunik() {
String term = "";
        String[] hasilStemming;
        for (int i = 0; i < data.length; i++) {
          Tabel6.setValueAt(String.valueOf(jTabel6.getValueAt(i,
1)).replaceAll("//", " "), i, 1);
            hasilStemming = String.valueOf(jTabel6.getValueAt(i,
1)).trim().split("\\s+");

jTabel6.setValueAt(String.valueOf(jTabel6.getValueAt(i,
1)).replaceAll("\\s+", " // "), i, 1);
            for (int j = 0; j < hasilStemming.length; j++) {
                term = term + (hasilStemming[j] + " ");
            }
        }
        String[] termarray = term.split("\\s+");
        71atrix71 trigger = false;
        for (int i = 0; i < termarray.length; i++) {
            if (i == 0) {
                hasilterm.add(termarray[i]);
            }
            for (int j = 0; j < hasilterm.size(); j++) {
                if      (!termarray[i].equalsIgnoreCase((String)
hasilterm.get(j))) {
                    trigger = true;
                } else {
                    trigger = false;
                    break;
                }
                if (j == (hasilterm.size() - 1) && trigger ==
true) {
                    hasilterm.add(termarray[i]);
                }
            }
        }
    }
```

Kode Program 5.6 pembentukan*term*unik

5.4.2 Fungsi rawTF

Fungsi **rawTF** adalah proses perhitungan banyaknya *term* yang muncul pada setiap dokumen. Pada kode program ditunjukan dengan membandingkan hasil proses *term* unik pada fungsi sebelumnya dengan hasil *Stemming* pada proses sebelumnya. Potongan program fungsi **rawTF** ditunjukan pada Kode Program 5.7.

```
public void rawTF() {
        pembentukantermunik();
        header2 = new String[data.length + 1];
```

```
        for (int i = 0; i < (data.length + 1); i++) {
            if (i == 0) {
                header2[0] = "Term";
            } else {
                header2[i] = "Dok " + I;
            }
        }
        DefaultTabelModel        tabelModel7        =        new
DefaultTabelModel(header2, hasilterm.size());
        jTabel7.setModel(tabelModel7);
        jTabel7.setAutoResizeMode(0);
        TabelColumnModel              columnModel7              =
jTabel7.getColumnModel();
        columnModel7.getColumn(0).setPreferredWidth(90);
        for (int i = 0; i < data.length; i++) {
            columnModel7.getColumn(i + 1).setPreferredWidth(60);

        }
        for (int i = 0; i < hasilterm.size(); i++) {
            jTabel7.setValueAt(hasilterm.get(i), i, 0);
        }
        int jumlahterm = 0;
        String[] hasilStemming2;
        for (int i = 0; i < data.length; i++) {
        jTabel6.setValueAt(String.valueOf(jTabel6.getValueAt(i,
1)).replaceAll("//", " "), i, 1);
            hasilStemming2                                    =
String.valueOf(jTabel6.getValueAt(i,   1)).trim().split("\\s+");
jTabel6.setValueAt(String.valueOf(jTabel6.getValueAt(i,
1)).replaceAll("\\s+", " // "), i, 1);
            for (int j = 0; j < hasilterm.size(); j++) {
                for (int k = 0; k < hasilStemming2.length; k++)
{
                    if
(hasilStemming2[k].equalsIgnoreCase(String.valueOf(hasilterm.get
(j)))) {
                        jumlahterm++;
                    }
                }
                jTabel7.setValueAt(jumlahterm, j, i + 1);
                jumlahterm = 0;
            }
        }
    }
```

Kode Program 5.7 rawTF

5.4.3 Fungsi pembobotanTF

Fungsi **pembobotanTF** adalah proses untuk menghitung pembobotan pada hasil *term* frekuensi. Proses perhitungan bobot pada setiap *term* dilakukan dengan menggunakan Persamaan 2.1. Potongan program fungsi **pembobotanTF** ditunjukan pada Kode Program 5.8.

```
public void pembobotanTF() {
        DefaultTabelModel        tabelModel8        =        new
DefaultTabelModel(header2, hasilterm.size());
        jTabel8.setModel(tabelModel8);
        jTabel8.setAutoResizeMode(0);
```

```
        TabelColumnModel                columnModel8                =
jTabel8.getColumnModel();
        columnModel8.getColumn(0).setPreferredWidth(90);
        for (int i = 0; i < data.length; i++) {
            columnModel8.getColumn(i + 1).setPreferredWidth(60);
        }
        for (int i = 0; i < hasilterm.size(); i++) {
            jTabel8.setValueAt(hasilterm.get(i), I, 0);
            for (int j = 0; j < data.length; j++) {
                if
(Double.valueOf(String.valueOf(jTabel7.getValueAt(i, j + 1))) ==
0) {
                    jTabel8.setValueAt(0, i, j + 1);
                } else {
                    jTabel8.setValueAt((1                          +
Math.log10(Double.valueOf(String.valueOf(jTabel7.getValueAt(i, j
+ 1))))), i, j + 1);
                }
            }
        }
    }
```

Kode Program 5.8 pembobotanTF

5.4.4 Fungsi perhitunganDF

Fungsi **perhitunganDF** adalah proses menghitung banyaknya jumlah dokumen yang memiliki kata pada dokumen. Potongan program fungsi **perhitunganDF** ditunjukan pada Kode Program 5.9.

```
public void perhitunganDF() {
        header3 = new String[data.length + 2];
        for (int i = 0; i < (data.length + 2); i++) {
            if (i == 0) {
                header3[0] = "Term";
            } else if (i == 1) {
                header3[1] = "DF";
            } else {
                header3[i] = "Dok " + (i - 1);
            }
        }
        DefaultTabelModel        tabelModel9        =        new
DefaultTabelModel(header3, hasilterm.size());
        jTabel9.setModel(tabelModel9);
        jTabel9.setAutoResizeMode(0);
        TabelColumnModel                columnModel9                =
jTabel9.getColumnModel();
        columnModel9.getColumn(0).setPreferredWidth(90);
        columnModel9.getColumn(1).setPreferredWidth(60);
        for (int i = 0; i < data.length; i++) {
            columnModel9.getColumn(i + 2).setPreferredWidth(60);
        }
        for (int i = 0; i < hasilterm.size(); i++) {
            jTabel9.setValueAt(hasilterm.get(i), i, 0);
        }
        for (int i = 0; i < data.length; i++) {
            for (int j = 0; j < hasilterm.size(); j++) {
                jTabel9.setValueAt(jTabel8.getValueAt(j, i + 1),
j, (i + 2));
```

```
            }
        }
        int df = 0;
        for (int j = 0; j < hasilterm.size(); j++) {
            for (int i = 0; i < data.length; i++) {
                if
(Double.valueOf(String.valueOf(jTabel8.getValueAt(j, i + 1))) !=
0) {
                    df++;
                }
            }
            jTabel9.setValueAt(df, j, 1);
            df = 0;
        }
    }
```

Kode Program 5.9 perhitunganDF

5.4.5 Fungsi pembobotanIDF

Fungsi **pembobotanIDF** adalah proses pembobotan yang mengukur seberapa pentingnya sebuah kata dalam dokumen dilihat dari keseluruhan dokumen secara global. Perhitungan bobot IDF menggunakan Persamaan 2.2. Potongan program fungsi **pembobotanIDF** ditunjukan pada Kode Program 5.10

```
public void pembobotanIDF() {
        header4 = new String[data.length + 2];
        for (int i = 0; i < (data.length + 2); i++) {
            if (i == 0) {
                header4[0] = "Term";
            } else if (i == 1) {
                header4[1] = "IDF";
            } else {
                header4[i] = "Dok " + (i - 1);
            }
        }
        DefaultTabelModel tabelModel10 = new
DefaultTabelModel(header4, hasilterm.size());
        jTabel10.setModel(tabelModel10);
        jTabel10.setAutoResizeMode(0);
        TabelColumnModel columnModel10 =
jTabel10.getColumnModel();

        columnModel10.getColumn(0).setPreferredWidth(90);
        columnModel10.getColumn(1).setPreferredWidth(60);

        for (int i = 0; i < data.length; i++) {
            columnModel10.getColumn(i +
2).setPreferredWidth(60);
        }

        for (int i = 0; i < hasilterm.size(); i++) {
            jTabel10.setValueAt(hasilterm.get(i), i, 0);
            jTabel10.setValueAt(Math.log10(data.length /
Double.valueOf(String.valueOf(jTabel9.getValueAt(i, 1)))), i,
1);
        }
        for (int i = 0; i < data.length; i++) {
            for (int j = 0; j < hasilterm.size(); j++) {
```

```
                jTabel10.setValueAt(jTabel9.getValueAt(j,   i   +
2), j, (i + 2));
            }
        }
    }
```

Kode Program 5.10 pembobotanIDF

5.4.6 Fungsi pembobotanTFIDF

Fungsi **pembobotanTFIDF** adalah proses pembobotan dokumen ke dalam model ruang *matrix*. Proses pembobotan pada TFIDF diimplementasikan dengan mengkalikan nilai hasil TF dan hasil IDF. Fungsi **pembobotanTFIDF** ini menggunakan Persamaan 2.3. Potongan program fungsi **pembobotanTFIDF** ditunjukan pada Kode Program 5.11.

```
public void pembobotanTFIDF() {

        DefaultTabelModel        tabelModel11        =        new
DefaultTabelModel(header2, hasilterm.size());
        jTabel11.setModel(tabelModel11);
        jTabel11.setAutoResizeMode(0);
        TabelColumnModel              columnModel11              =
jTabel11.getColumnModel();
        columnModel11.getColumn(0).setPreferredWidth(90);
        for (int i = 0; i < data.length; i++) {
          columnModel11.getColumn(i + 1).setPreferredWidth(60);
        }

        for (int i = 0; i < hasilterm.size(); i++) {
            jTabel11.setValueAt(hasilterm.get(i), i, 0);
        }


        for (int i = 0; i < data.length; i++) {
            for (int j = 0; j < hasilterm.size(); j++) {
                double tfidf = 0;
 tfidf = Double.valueOf(String.valueOf(jTabel10.getValueAt(j, i +
2)))    *    Double.valueOf(String.valueOf(jTabel10.getValueAt(j,
1)));
                jTabel11.setValueAt(tfidf, j, i + 1);

            }
        }
    }
```

Kode Program 5.11 pembobotanTFIDF

5.5 Singular Value Decomposition

Proses *Singular Value Decomposition* digunakan untuk menurunkan dimensi matrik (reduksi matrik). Pada proses ini, matrik hasil proses TFIDF akan dipecah menjadi 3 *sub* matrik. *Sub* matrik terdiri dari matrik U, matrik S dan matrik V^T. Proses *Singular Value Decomposition* diterapkan dengan menggunakan fungsi *SVD*

5.5.1 Fungsi SVD

Fungsi **SVD** adalah proses pemecahan matrik TFIDF menjadi *sub* matrik U, matrik S dan matrik V^T. Pada implementasi sistem, pemecahan matrik tersebut menggunakan *library* yang telah disediakan yaitu *library jama*. *Library jama* menyediakan beberapa paket fungsi yang bisa digunakan, salah satunya adalah *Singular Value Decomposition*. Penerapan proses fungsi **SVD** adalah dari Persamaan 2.4. Potongan program fungsi **SVD** ditunjukan pada Kode Program 5.12.

```
public void SVD() {
        matrixtfidf = new double[hasilterm.size()][data.length];
        for (int i = 0; i < data.length; i++) {
            for (int j = 0; j < hasilterm.size(); j++) {
                matrixtfidf[j][i]                                    =
Double.valueOf(String.valueOf(jTabel11.getValueAt(j, i + 1)));
            }
        }
        Matrix SVD = new Matrix(matrixtfidf);

        SingularValueDecomposition s = SVD.svd();

        Matrix U = s.getU();
        Matrix = U.getArray();

        DefaultTabelModel      tabelModel12       =       new
DefaultTabelModel(76atrix.length, 76atrix[0].length);
        jTabel12.setModel(tabelModel12);
        jTabel12.setAutoResizeMode(0);
        TabelColumnModel                columnModel12                =
jTabel12.getColumnModel();
        jTabel12.getTabelHeader().setUI(null);
        for (int i = 0; i < 76atrix.length; i++) {  //baris
            for (int j = 0; j < 76atrix[0].length; j++) { //kolom
                jTabel12.setValueAt(76atrix[i][j], i, j);
            }
        }

        Matrix S = s.getS();
        matrixS = S.getArray();

        DefaultTabelModel       tabelModel13       =       new
DefaultTabelModel(matrixS.length, matrixS[0].length);
        jTabel13.setModel(tabelModel13);
        jTabel13.setAutoResizeMode(0);
        TabelColumnModel              columnModel13              =
jTabel13.getColumnModel();
        jTabel13.getTabelHeader().setUI(null);
        for (int i = 0; i < matrixS.length; i++) {  //baris
            for (int j = 0; j < matrixS[0].length; j++) { //kolom
                jTabel13.setValueAt(matrixS[i][j], i, j);
            }
        }
        Matrix V = s.getV();
        Matrix Vt = V.transpose();
        matrixVt = Vt.getArray();
```

```
        DefaultTabelModel        tabelModel14        =        new
DefaultTabelModel(matrixVt.length, matrixVt[0].length);
        jTabel14.setModel(tabelModel14);
        jTabel14.setAutoResizeMode(0);
        TabelColumnModel              columnModel14              =
jTabel14.getColumnModel();
        jTabel14.getTabelHeader().setUI(null);
        for (int i = 0; i < matrixVt.length; i++) {  //baris
            for (int j = 0; j < matrixVt[0].length; j++) { //kolom
                jTabel14.setValueAt(matrixVt[i][j], i, j);
            }
        }
    }
```

Kode Program 5.12 SVD

5.6 Latent Semantic Indexing

Latent Semantic Indexing (LSI) digunakan untuk mereduksi matrik hasil pemecahan *Singular Value Decomposition.* Adapun pada penerapannya, LSI diimplementasikan pada fungsi *reducematrixLSI*.

5.6.1 Fungsi reducematrixLSI

Fungsi **reducematrixLSI** adalah proses pemecahan matrik hasil dari proses fungsi SVD. Pada proses ini, matrik yang digunakan untuk membentuk matrik LSI adalah matrik S dan matrik V^T seperti pada Persamaan 2.6. Pertama, masukan *rank* pada user akan diproses untuk memotong matrik S dan matrik V^T. Kedua, hasil pemotongan matrik S dan matrik V^T dikalikan untuk mendapatkan matrik LSI. Potongan program fungsi **reducematrixLSI** ditunjukan pada Kode Program 5.13.

```
public void reducematrixLSI() {
        if (jComboBox1.getSelectedIndex() == 0) {
            rank = 100;
        } else if (jComboBox1.getSelectedIndex() == 1) {
            rank = 95;
        } else if (jComboBox1.getSelectedIndex() == 2) {
            rank = 90;
        } else if (jComboBox1.getSelectedIndex() == 3) {
            rank = 85;
        } else if (jComboBox1.getSelectedIndex() == 4) {
            rank = 80;
        } else if (jComboBox1.getSelectedIndex() == 5) {
            rank = 75;
        } else if (jComboBox1.getSelectedIndex() == 6) {
            rank = 70;
        } else if (jComboBox1.getSelectedIndex() == 7) {
            rank = 65;
        } else if (jComboBox1.getSelectedIndex() == 8) {
            rank = 60;
        } else if (jComboBox1.getSelectedIndex() == 9) {
            rank = 55;
        } else if (jComboBox1.getSelectedIndex() == 10) {
            rank = 50;
        } else if (jComboBox1.getSelectedIndex() == 11) {
            rank = 45;
        } else if (jComboBox1.getSelectedIndex() == 12) {
```

```
            rank = 40;
        } else if (jComboBox1.getSelectedIndex() == 13) {
            rank = 35;
        } else if (jComboBox1.getSelectedIndex() == 14) {
            rank = 30;
        } else if (jComboBox1.getSelectedIndex() == 15) {
            rank = 25;
        } else if (jComboBox1.getSelectedIndex() == 16) {
            rank = 20;
        } else if (jComboBox1.getSelectedIndex() == 17) {
            rank = 15;
        } else if (jComboBox1.getSelectedIndex() == 18) {
            rank = 10;
        } else if (jComboBox1.getSelectedIndex() == 19) {
            rank = 5;
        }

        double barisVT = matrixVt.length * (rank / 100);
        int brVT = (int) barisVT;

        DefaultTabelModel tabelModel15 = new
DefaultTabelModel(brVT, matrixVt[0].length);
        jTabel15.setModel(tabelModel15);
        jTabel15.setAutoResizeMode(0);
        TabelColumnModel columnModel15 =
jTabel15.getColumnModel();
        jTabel15.getTabelHeader().setUI(null);
        DefaultTabelModel tabelModel17 = new
DefaultTabelModel(matrixVt[0].length, brVT);
        jTabel17.setModel(tabelModel17);
        jTabel17.setAutoResizeMode(0);
        TabelColumnModel columnModel17 =
jTabel17.getColumnModel();
        jTabel17.getTabelHeader().setUI(null);

        for (int i = 0; i < brVT; i++) {  //baris
            for (int j = 0; j < matrixVt[0].length; j++) { //kolom
                jTabel15.setValueAt(matrixVt[i][j], i, j);
                jTabel17.setValueAt(matrixVt[i][j], j, i);
            }
        }

        double barisS = matrixS.length * (rank / 100);
        int brS = (int) barisS;
        DefaultTabelModel tabelModel16 = new
DefaultTabelModel(brS, brS);
        jTabel16.setModel(tabelModel16);
        jTabel16.setAutoResizeMode(0);
        TabelColumnModel columnModel16 =
jTabel16.getColumnModel();
        jTabel16.getTabelHeader().setUI(null);
        for (int i = 0; i < brS; i++) {  //baris
            for (int j = 0; j < brS; j++) { //kolom
                jTabel16.setValueAt(matrixS[i][j], i, j);
            }
        }
        V = Matrix.constructWithCopy(matrixVt);
        V = V.getMatrix(0, brVT - 1, 0, matrixVt[0].length - 1);
        V = V.transpose();
```

```
        matrixVbaru = V.getArray();
        System.out.println("matrix V");
        V.print(12, 4);

        S = Matrix.constructWithCopy(matrixS);
        S = S.getMatrix(0, brS - 1, 0, brS - 1);
        matrixSbaru = S.getArray();
        System.out.println("matrix S");
        S.print(12, 4);
        LSI = V.times(S);
        System.out.println("matrix LSI");
        LSI.print(12, 4);
        matrixLSI = LSI.transpose().getArray(); // mempermudah
menilai perdokumen sehingga ditranspose
        header5 = new String[data.length];
        for (int i = 0; i < (data.length); i++) {
            header5[i] = "Dok " + (i + 1);
        }
        DefaultTabelModel tabelModel18 = new
DefaultTabelModel(header5, matrixLSI.length);
        jTabel18.setModel(tabelModel18);
        jTabel18.setAutoResizeMode(0);
        TabelColumnModel columnModel18 =
jTabel18.getColumnModel();
        for (int i = 0; i < matrixLSI.length; i++) {  //baris
            for (int j = 0; j < matrixLSI[0].length; j++) {
                jTabel18.setValueAt(matrixLSI[i][j], i, j);
            }
        }
    }
```

Kode Program 5.13 reducematrixLSI

5.7 Cosine Similarity

Cosine similarity digunakan untuk mengukur kemiripan antar dokumen. Pada proses ini *cosine similarity* mengubah koleksi dokumen ke dalam bentuk matrik. Operasi yang dilakukan pada model ruang vektor adalah *dot product*. Beberapa dokumen bisa diartikan sebagai sekumpulan vektor pada ruang vektor, yang berada pada satu sumbu untuk setiap *term.* Proses *cosine similarity* diterapkan dengan fungsi *hitungcosinesimilarity*.

5.7.1 Fungsi hitungcosinesimilarity

Fungsi **hitungcosinesimilarity** adalah proses perhitungan kemiripan antar dokumen setelah proses menghitung matrik *Latent Semantic Indexing*. Matrik LSI yang merepresentasikan dokumen dan *term* akan dihitung kemiripan setiap dokumennya menggunakan Persamaan 2.6. potongan program fungsi **hitungcosinesimilarity** ditunjukan pada Kode Program 5.14.

```
  public void hitungcosinesimilarity() {
        header6 = new String[matrixLSI[0].length + 1];
        for (int i = 0; i < (matrixLSI[0].length + 1); i++) {
            if (i == 0) {
                header6[0] = "Cosine";
            } else {
                header6[i] = "Dok " + i;
```

```
            }
        }
        DefaultTabelModel        tabelModel19        =        new
DefaultTabelModel(header6, matrixLSI[0].length);
        jTabel19.setModel(tabelModel19);
        jTabel19.setAutoResizeMode(0);
        TabelColumnModel             columnModel19             =
jTabel19.getColumnModel();
        for (int i = 0; i < matrixLSI[0].length; i++) {
            jTabel19.setValueAt("Dok " + (i + 1), i, 0);
        }
        matrikcluster                    =                    new
double[matrixLSI[0].length][matrixLSI[0].length];

        for (int a = 0; a < matrixLSI[0].length; a++) {  //baris
            for (int b = 0; b < matrixLSI[0].length; b++) {
                double dotProduct = 0.0;
                double penyebutA = 0.0;
                double PenyebutB = 0.0;
                for (int c = 0; c < matrixLSI.length; c++) {
                    dotProduct      +=      matrixLSI[c][a]      *
matrixLSI[c][b];
                    penyebutA += Math.pow(matrixLSI[c][a], 2);
                    PenyebutB += Math.pow(matrixLSI[c][b], 2);
                }
                jTabel19.setValueAt(dotProduct                    /
(Math.sqrt(penyebutA) * Math.sqrt(PenyebutB)), a, b + 1);
                matrikcluster[a][b]      =      dotProduct      /
(Math.sqrt(penyebutA) * Math.sqrt(PenyebutB));
            }
        }
        for (int i = 0; i < matrikcluster.length; i++) {
            matriklabelcluster.add(I + 1);
        }
        matrikcluster2 = matrikcluster;
    }
```

Kode Program 5.14 hitungcosinesimilarity

5.8 *Clustering* UPGMA

Clustering UPGMA adalah algoritme pengelompokan hierarchical *cluster*ing. UPGMA menggunakan pengukuran rata-rata dalam proses perhitungan jarak antar pasangan data yang telah dikelompokkan dengan jarak data yang lainnya. Beberapa fungsi untuk menerapkan algoritme *clustering* ini terdiri dari fungsi *caridekat*, fungsi *update*, fungsi *updatearray*, fungsi *jarak*, dan fungsi *ratarata*.

5.8.1 Fungsi caridekat

Fungsi **caridekat** adalah proses awal pada pembentukan *cluster* baru dengan metode UPGMA *clustering*. Pada fungsi ini akan dicari jarak 2 dokumen mana yang memiliki nilai *cosine similarity* terbesar. Kedua dokumen tersebut akan disimpan pada variabel *dekat1* dan *dekat2*. Selanjutnya pada fungsi ini akan memanggil fungsi lain dengan menggunakan parameter variabel *dekat1* dan *dekat2* untuk mengupdate matrik *cosine similarity* pada proses sebelumnya. Potongan program fungsi **caridekat** ditunjukan pada Kode Program 5.15.

```
public static void caridekat(double tempmatrik[][]) {

        double temp = -2;
        for (int i = 0; i < tempmatrik.length; i++) {
            for (int j = 0; j < tempmatrik[0].length; j++) {
                if (tempmatrik[i][j] > temp && I != j) {
                    temp = tempmatrik[i][j];
                    dekat1 = i + 1;
                    dekat2 = j + 1;
                }
            }
        }

        nilaidekat = temp;
        update(dekat1, dekat2);
        matrikcluster = updatearray(matrikcluster);
        ratarata(matrikcluster);
    }
```

Kode Program 5.15 caridekat

5.8.2 Fungsi update

Fungsi **update** adalah proses untuk menghapus dua dokumen yang sudah ditentukan sebagai dokumen yang memiliki jarak terdekat atau nilai *cosine similarity* terbesar. Setelah dua dokumen beserta nilai jarak dengan dokumen lainnya dihapuskan, langkah selanjutnya menambahkan satu dokumen baru diawal penyimpanan dokumen *lingkedlist*. Di mana dokumen baru tersebut diartikan sebagai *cluster* baru hasil penggabungan dari kedua dokumen yang memiliki jarak terdekat. Potongan program fungsi **update** ditunjukan pada Kode Program 5.16.

```
public static void update(int tempdekat1, int tempdekat2) {
        Object atempdekat1 = matriklabelcluster.get(dekat1 - 1);
        Object atempdekat2 = matriklabelcluster.get(dekat2 - 1);
        String temp1 = "";
        String temp2 = "";
        String[] smp1 = String.valueOf(atempdekat1).split(",");
        String[] smp2 = String.valueOf(atempdekat2).split(",");
        for (int i = 0; i < smp1.length; i++) {
            temp1 = temp1.concat(smp1[i]); }
        for (int i = 0; i < smp2.length; i++) {
            temp2 = temp2.concat(smp2[i]);
        }
        matriklabelcluster.addFirst("("                          +
String.valueOf(atempdekat1).concat(","                           +
String.valueOf(atempdekat2)) + ")");
        String simpandat1 = "", simpandat2 = "";
        int simpanhapus = 0;
        for (int i = 0; i < matriklabelcluster.size(); i++) {
            String                    dataa[]                    =
String.valueOf(matriklabelcluster.get(i)).split(",");
            for (int j = 0; j < dataa.length; j++) {
                simpandat1 = simpandat1.concat(dataa[j]);
            }
            if (simpandat1.equals(String.valueOf(temp1))) {
```

```
                simpanhapus = I;
                break;
            }
            simpandat1 = "";       }
        dekat11                                                    =
String.valueOf(matriklabelcluster.get(simpanhapus));
        matriklabelcluster.remove(simpanhapus);
        int simpanhapus2 = 0;
        for (int i = 0; i < matriklabelcluster.size(); i++) {
            String                    dataa[]                      =
String.valueOf(matriklabelcluster.get(i)).split(",");
            for (int j = 0; j < dataa.length; j++) {
                simpandat2 = simpandat2.concat(dataa[j]);
            }
            if (simpandat2.equals(String.valueOf(temp2))) {
                simpanhapus2 = i;
            }
            simpandat2 = "";}
               dekat22                                             =
String.valueOf(matriklabelcluster.get(simpanhapus2));
        matriklabelcluster.remove(simpanhapus2);
    }
```

Kode Program 5.16 update

5.8.3 Fungsi updatearray

Fungsi **updatearray** adalah proses untuk membuat array baru tempat penyimpanan nilai jarak setiap dokumen. Pada setiap iterasinya, jumlah baris dan kolom akan berkurang satu. Hal tersebut dikarenakan pada setiap iterasinya dua dokumen yang memiliki jarak terdekat akan digabungkan menjadi satu *cluster*. Fungsi **updatearray** memiliki nilai kembalian berupa array dua dimensi, array tersebut adalah bentuk matrik dari hasil pembentukan *cluster* baru. Potongan program fungsi **updatearray** ditunjukan pada Kode Program 5.17.

```
public static double[][] updatearray(double matriktemp[][]) {
        double  nilaibaru[][]  =  new  double[matriktemp.length  -
1][matriktemp.length - 1];
        int ii = 0; jj = 0;
        for (int i = 0; i < matriktemp.length; i++) {
            if (i == dekat1 - 1 || I == dekat2 - 1) {
                continue;                }
            for (int j = 0; j < matriktemp[0].length; j++) {
                if (j == dekat2 - 1 || j == dekat1 - 1) {
                    continue;
                }
                nilaibaru[ii + 1][jj + 1] = matriktemp[i][j];
                jj++;}
                        jj = 0;
            ii++; }
               return nilaibaru;}
```

Kode Program 5.17 updatearray

5.8.4 Fungsi jarak

Fungsi **jarak** adalah proses untuk menjumlahkan nilai *similarity* atau kemiripan pada *cluster* baru dengan *cluster* lainnya. Fungsi **jarak** mengembalikan nilai

variabel *double* bernama *hasil*. Potongan program fungsi **jarak** ditunjukan pada Kode Program 5.18.

```
public static double jarak(String[] Matrika, String[] Matrikb) {

        double hasil = 0;
        int tempa1 = 0;
        int tempb1 = 0;
        for (int i = 0; i < Matrika.length; i++) {
            tempa1 = Integer.valueOf(Matrika[i]);
            for (int j = 0; j < Matrikb.length; j++) {
                tempb1 = Integer.valueOf(Matrikb[j]);
                hasil = hasil + matrikcluster2[tempa1 - 1][tempb1
- 1];
            }
        }

        return hasil;

    }
```

Kode Program 5.18 jarak

5.8.5 Fungsi ratarata

Fungsi **ratarata** adalah proses menentukan jarak *cluster* baru dengan *cluster* lainnya pada penerapan algoritme *clustering* UPGMA. Fungsi **ratarata** menggunakan Persamaan 2.7 dalam menghitung nilai jarak tersebut. Potongan program fungsi **ratarata** ditunjukan pada Kode Program 5.19.

```
public static void ratarata(double temp[][]) {

        String                    simpann                    =
String.valueOf(matriklabelcluster.get(0));

        simpann = simpann.replace("(", "");
        simpann = simpann.replace(")", "");
        String arrsimpann[] = simpann.split(",");
        double hasil = 0;
        temp[0][0] = 1;

        for (int i = 1; i < matrillabelcluster.size(); i++) {
            String                  simpan2                  =
String.valueOf(matriklabelcluster.get(i));
            simpan2 = simpan2.replace("(", "");
            simpan2 = simpan2.replace(")", "");
            String smp[] = simpan2.split(",");

            hasil = jarak(arrsimpann, smp);

            temp[i][0]   =   hasil   /   (arrsimpann.length   *
smp.length);
            temp[0][i]   =   hasil   /   (arrsimpann.length   *
smp.length);
            hasil = 0;
```

```
        }
}
```

Kode Program 5.19 ratarata

5.9 Cophenetic Correlation Coefficient

Cophenetic Correlation Coefficient merupakan koefisien korelasi antara matrik jarak dan matrik jarak *dendrogram* atau matriks *cophenetic* digunakan untuk melihat kualitas pada *cluster*.

5.9.1 Fungsi *copheneticcorrelation*

Fungsi ***cophenetic*corralation** digunakan untuk menghitung matrik *cophenetic* atau matrik jarak *dendrogram* dengan matrik jarak *cosine similarity* pada dokumen. Hasil dari matrik *cophenetic* dan matrik jarak akan digunakan untuk menghitung nilai korelasi dengan Persamaan 2.8. Potongan program fungsi ***copheneticcorrelation*** ditunjukan pada Kode Program 5.23.

```
public void copheneticcorrelation() {
        double[] copheneticmatrix = new
double[((jTabel22.getRowCount() * (jTabel22.getColumnCount() -
1)) - jTabel22.getRowCount()) / 2];
        double[] distancematrix = new
double[((jTabel19.getRowCount() * (jTabel19.getColumnCount() -
1)) - jTabel19.getRowCount()) / 2];

        int counter = 1;
        int countercopheneticmatrix = 0;
        for (int i = 0; i < jTabel22.getRowCount(); i++) {
            for (int j = counter; j < jTabel22.getColumnCount()
- 1; j++) {
                copheneticmatrix[countercopheneticmatrix] =
Double.valueOf(String.valueOf(jTabel22.getValueAt(i, j + 1)));
                countercopheneticmatrix++;
            }
            counter++;
        }

        int counter2 = 1;
        int counterdistancematrix = 0;
        for (int i = 0; i < jTabel19.getRowCount(); i++) {
            for (int j = counter2; j < jTabel19.getColumnCount()
- 1; j++) {
                distancematrix[counterdistancematrix] =
Double.valueOf(String.valueOf(jTabel19.getValueAt(i, j + 1)));
                counterdistancematrix++;
            }
            counter2++;
        }

        double rataratacophenetic = 0;
        double rataratajarak = 0;

        for (int i = 0; i < copheneticmatrix.length; i++) {
            rataratacophenetic = rataratacophenetic +
copheneticmatrix[i];
```

```
        }
        rataratacophenetic = rataratacophenetic /
copheneticmatrix.length;

        for (int i = 0; i < distancematrix.length; i++) {
            rataratajarak = rataratajarak + distancematrix[i];
        }
        rataratajarak = rataratajarak / distancematrix.length;

        double akhir = 0;
        double atas = 0;
        double bawahkanan = 0;
        double bawahkiri = 0;
        double bawah = 0;
        for (int i = 0; i < copheneticmatrix.length; i++) {
            atas = atas + ((copheneticmatrix[i] -
rataratacophenetic) * (distancematrix[i] - rataratajarak));
        }

        //jarak
        for (int i = 0; i < distancematrix.length; i++) {
            bawahkiri = bawahkiri + Math.pow(distancematrix[i] -
rataratajarak, 2);
        }
        //cophenetic
        for (int i = 0; i < copheneticmatrix.length; i++) {
            bawahkanan = bawahkanan +
Math.pow(copheneticmatrix[i] - rataratacophenetic, 2);
        }
        bawah = Math.sqrt(bawahkiri * bawahkanan);
        akhir = atas / bawah;

        System.out.println("nilai cophenetic = " + akhir);
        jLabel18.setText("" + akhir);
}
```

Kode program 5.20 *Cophenetic Correlation Coefficient*

5.10 Silhouette Coefficient

Silhouette Coefficient adalah metode evaluasi hasil *clustering* yang sesuai pada penerapan metode *hierarchical clustering* UPGMA. Metode ini merupakan metode yang digunakan untuk mengukur kualitas dan kekuatan suatu *cluster*, di mana pada *cluster* tersebut dapat diukur seberapa baik suatu *object* sudah ditempatkan pada *cluster* yang sesuai.

5.10.1 Fungsi evaluasi*Silhouette*

Fungsi **evaluasi*Silhouette*** adalah fungsi untuk mencari nilai evaluasi menggunakan rumus Persamaan 2.11. Fungsi ini memanggil fungsi lainnya yaitu fungsi **hitungai** dan fungsi **hitungbi**. Hasil dari memanggil kedua fungsi tersebut akan digunakan untuk menghitung nilai *si*. Nilai *si* akan dihitung rata-ratanya dan akan dijadikan sebagai pengembalian saat memanggil fungsi **evaluasi*Silhouette***. Potongan program fungsi **evaluasi*Silhouette*** ditunjukan pada Kode Program 5.20.

```
public double evaluasiSilhouette(LinkedList ab, double[][]
cosinematrik) {
```

```
        double[] si = new double[simpanmatrikcosine.length];
        double averagesi = 0;
        String simpani = "";

        double[] nilaiai = new double[simpanmatrikcosine.length];
        double[] nilaibi = new double[simpanmatrikcosine.length];
        double[] ratasipercluster = new double[ab.size()];

        for (int i = 0; i < simpanmatrikcosine.length; i++) {

            for (int j = 0; j < ab.size(); j++) {
                String simpann = String.valueOf(ab.get(j));
                simpann = simpann.replace("(", "");
                simpann = simpann.replace(")", "");
                String arrsimpann[] = simpann.split(",");
                int untukberhenti = 0;
                for (int k = 0; k < arrsimpann.length; k++) {

                    if (arrsimpann[k].equals("" + (i + 1))) {
                        simpani = "" + (j + 1);
                        untukberhenti = 1;
                        break;
                    }
                }
                if (untukberhenti == 1) {
                    break;
                }

            }
//              untuk mencari nilai ai
            nilaiai[i] = hitungai(ab, simpani, i + 1);
            System.out.println("ai  " + (i + 1) + "  " +
hitungai(ab, simpani, i + 1));

//              untuk mencari nilai bi
            nilaibi[i] = hitungbi(ab, simpani, i + 1);
            System.out.println("bi  " + (i + 1) + "  " +
hitungbi(ab, simpani, i + 1));
        }

        for (int i = 0; i < si.length; i++) {
            if (nilaiai[i] == 0) {
                si[i] = 0;
            } else {
                if (nilaiai[i] > nilaibi[i]) {
                    si[i] = 1 - (nilaibi[i] / nilaiai[i]);
                } else if (nilaiai[i] == nilaibi[i]) {
                    si[i] = 1 - (nilaibi[i] / nilaiai[i]);
                } else if (nilaiai[i] < nilaibi[i]) {
                    si[i] = (nilaiai[i] / nilaibi[i]) - 1;
                }
            }
            jTabel21.setValueAt(si[i], triggerevaluasi, i + 2);
            System.out.println("nilai si " + si[i]);
        }

        //rata rata si per cluster
        int clusterke = 0;
        for (int i = 0; i < ab.size(); i++) {
```

```
            ratasipercluster[i] = 0;
            String simpann = String.valueOf(ab.get(i));
            simpann = simpann.replace("(", "");
            simpann = simpann.replace(")", "");
            String arrsimpann[] = simpann.split(",");

            for (int j = 0; j < arrsimpann.length; j++) {
                clusterke = Integer.valueOf(arrsimpann[j]);
                ratasipercluster[i]   =   ratasipercluster[i]   +
si[clusterke - 1];
            }
            ratasipercluster[i]    =    ratasipercluster[i]    /
arrsimpann.length;

        }

        for (int i = 0; i < ratasipercluster.length; i++) {
            averagesi = averagesi + ratasipercluster[i];
        }

        averagesi = averagesi / ratasipercluster.length;
        System.out.println("rata rata si : " + averagesi);

        return averagesi;
    }
```

Kode Program 5.21 evaluasi*Silhouette*

5.10.2 Fungsi hitungai

Fungsi **hitungai** adalah fungsi untuk menghitung jarak rata-rata *object I* dengan *object* lainnya dalam *cluster* yang sama. Fungsi ini menjumlahkan nilai jarak *object I* dengan *object* lainnya, kemudian hasil dari penjumlahan nilai *ai* akan dibagi dengan jumlah seluruh *object I* pada *cluster* yang sama dikurangi 1. Potongan program fungsi **hitungai** ditunjukan pada Kode Program 5.21.

```
public double hitungai(LinkedList ab, String simpani, int i) {
        double ai = 0;

        String                      simpann2                      =
String.valueOf(ab.get(Integer.valueOf(simpani) - 1));
        simpann2 = simpann2.replace("(", "");
        simpann2 = simpann2.replace(")", "");
        String arrsimpann2[] = simpann2.split(",");

        if (arrsimpann2.length == 1) {
            ai = 0;
            return ai;
        } else {
            for (int j = 0; j < arrsimpann2.length; j++) {
                if (Integer.valueOf(arrsimpann2[j]) == i) {
                    continue;
                }

                ai    =    ai    +    simpanmatrikcosine[I    -
1][Integer.valueOf(arrsimpann2[j]) - 1];

            }
            return ai / (arrsimpann2.length - 1);
```

```
        }
    }
```

Kode program 5.22 hitungai

5.10.3 Fungsi hitungbi

Fungsi **hitungbi** adalah fungsi untuk menghitung nilai maksimum rata-rata jarak *object I* dengan *object* lainnya pada *cluster* yang berbeda. Fungsi ini mengembalikan nilai *bi* yang dihitung seusai rumus Persamaan 2.10. Potongan program fungsi **hitungbi** ditunjukan pada Kode Program 5.22.

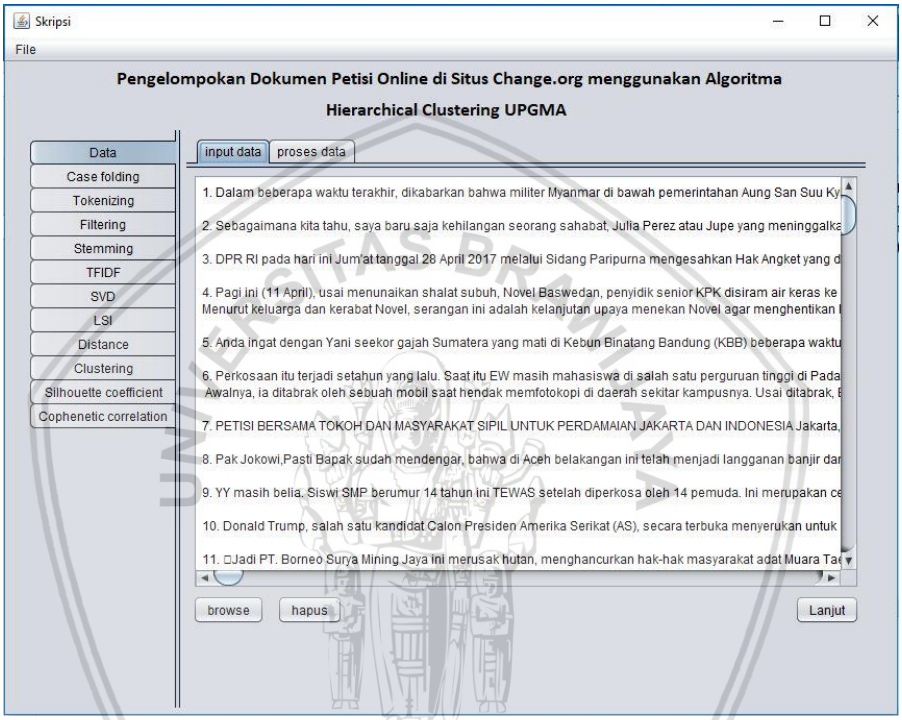
```
public double hitungbi(LinkedList ab, String simpani, int i) {
        double bi = 0;
        double[] tempbi = new double[ab.size() - 1];
        int trigger = 0;
        for (int j = 0; j < ab.size(); j++) {
            if (j == (Integer.valueOf(simpani) - 1)) {
                continue;
            } else {
                String simpann = String.valueOf(ab.get(j));
                simpann = simpann.replace("(", "");
                simpann = simpann.replace(")", "");
                String arrsimpann[] = simpann.split(",");
                for (int k = 0; k < arrsimpann.length; k++) {
                    bi = bi + simpanmatrikcosine[I -
1][Integer.valueOf(arrsimpann[k]) - 1]; }
                tempbi[trigger] = bi / arrsimpann.length;
                trigger++;
                bi = 0;
            }
        }
        if (ab.size() != 1) {
            bi = tempbi[0];
            for (int j = 0; j < tempbi.length; j++) {
                bi = Math.max(tempbi[j], bi);
            }
        }
        return bi;
    }
```

Kode program 5.23 hitungbi

5.11 Implementasi Antarmuka

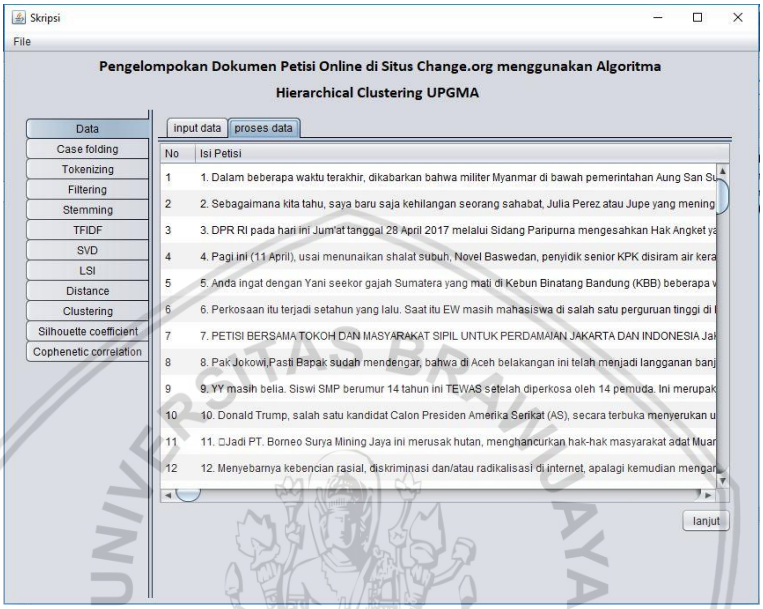
Pada *sub* bab implementasi antarmuka, akan diberikan tampilan hasil dari implementasi pengelompokan dokumen petisi *online* di situs change.org menggunakan algoritme *hierarchical clustering* UPGMA. Implementasi antarmuka ini terdiri dari 11 menu. Menu terdiri dari menu **Data**, menu **case folding**, menu **Tokenizing**, menu **Filtering**, menu **Stemming**, menu **TFIDF**, menu **SVD**, menu **LSI**, menu **cosine similarity**, menu **clustering** dan menu **Evaluasi**. Setiap menu tersebut didalamnya terdapat beberapa *sub* menu untuk proses lebih detail disetiap menunya.

Pada tahap awal di menu **data**, dilakukan pengambilan data petisi *online* dengan menekan tombol browse pada *sub* menu **input data.** Setelah menekan tombol *browse*, maka pengguna diharuskan untuk memasukkan file data yang diinginkan dan sistem akan menampilkan data tersebut pada *text area* yang sudah disediakan pada *sub* menu **input data**. Untuk lanjut pada proses selanjutnya , pengguna diperlukan menekan tombol lanjut pada bagian kanan bawah antarmuka sistem. Fungsi tombol tersebut adalah untuk menampilkan hasil *input* pengguna pada tabel secara detail. Halaman antarmuka **input data** ditunjukan pada Gambar 5.1.



Gambar 5.1 Implementasi Antarmuka *Input* Data

Pada halaman antarmuka dokumen hasil *input* akan ditampilkan pada tabel di*sub* menu **proses data**. Data akan disusun perdokumen yang diklasifikasikan berdasarkan jarak paragraf. Pada halaman antarmuka ini, data dokumen dijadikan sebagai bahan awal untuk proses *Text Preprocessing* dokumen di halaman antarmuka selanjutnya. Tabel pada *sub* menu **proses data** terdiri dari nomor dan isi petisi. Nomor diartikan sebagai dokumen. Halaman antarmuka **proses data** ditunjukan pada Gambar 5.2.



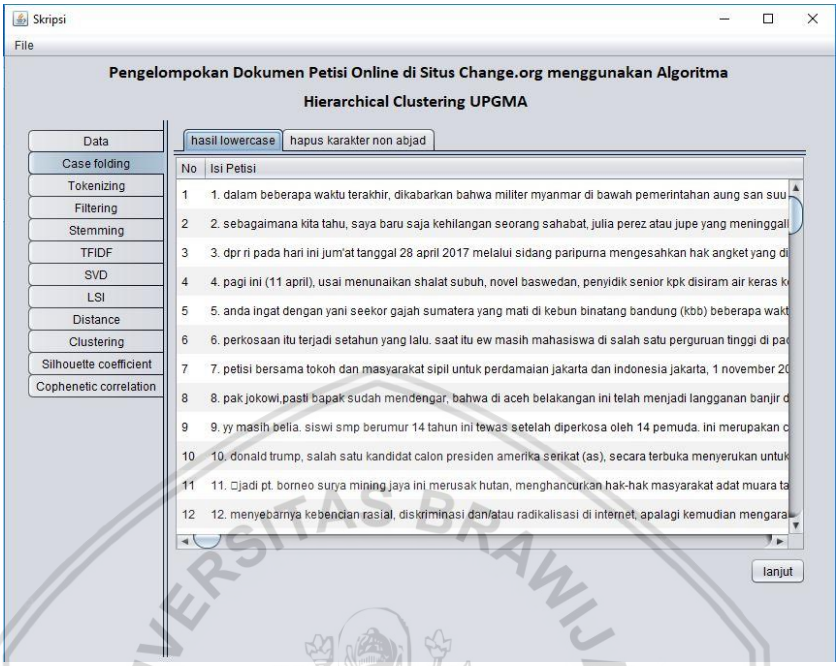
Gambar 5.2 Implementasi Antarmuka Proses Data

Pada menu **case folding** terdapat dua *sub* menu, yaitu **hasil lowercase** dan **hapus karakter non abjad**.

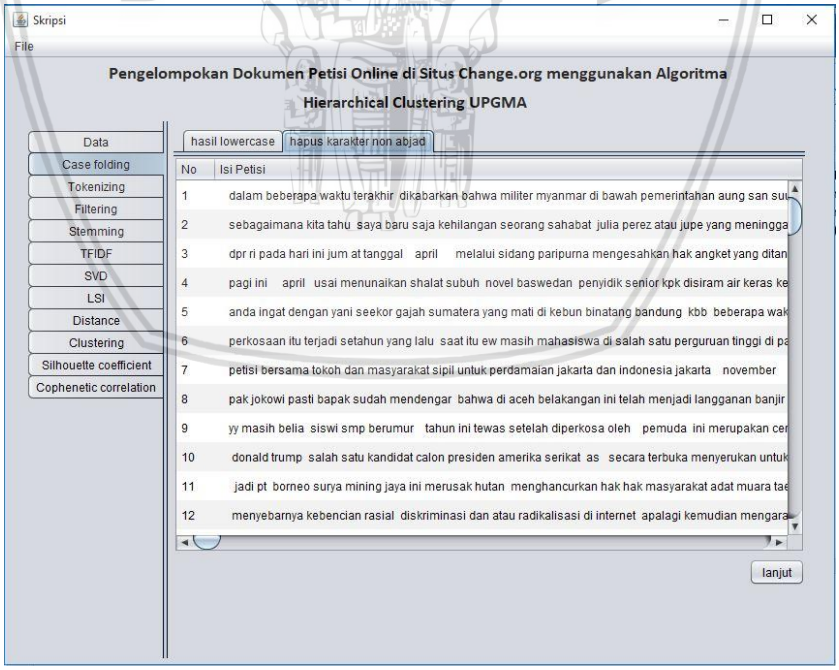
Pada halaman antarmuka *sub* menu **hasil lowercase**, data *input* pengguna ditampilkan kembali dalam bentuk tabel. Tabel pada *sub* menu **hasil lowercase** sudah dilakukan proses perubahan semua data dokumen yang memiliki huruf kapital diubah menjadi huruf kecil. Kolom pada tabel sama seperti halaman antarmuka sebelumnya, terdiri dari nomor yang menggambarkan dokumen dan isi petisi yang menggambarkan data dokumen yang diproses disetiap menunya. Pada halaman ini juga terdapat tombol lanjut yang digunakan untuk berpindah halaman dan memproses data dokumen pada halaman antarmuka selanjutnya yaitu *sub* menu **hapus karakter non abjad**. Halaman antarmuka **hasil *lowercase*** ditunjukan pada Gambar 5.3.

Sedangkan pada halaman antarmuka *sub* menu **hapus karakter non abjad**, data dokumen petisi yang telah melalui proses *lowercase* akan dicek disetiap karakternya apakah terdapat karakter selain huruf abjad. Jika terdapat karakter tersebut maka akan diganti dengan karakter spasi. Hasil proses penghapusan karakter non abjad ini ditampilkan pada tabel yang memiliki 2 kolom yaitu nomor dan isi petisi. Pada halaman ini juga terdapat tombol lanjut yang digunakan untuk

berpindah halaman dan memproses data dokumen pada halaman antarmuka selanjutnya yaitu menu **Tokenizing**. Halaman antarmuka **hapus karakter non abjad** ditunjukan pada Gambar 5.4.



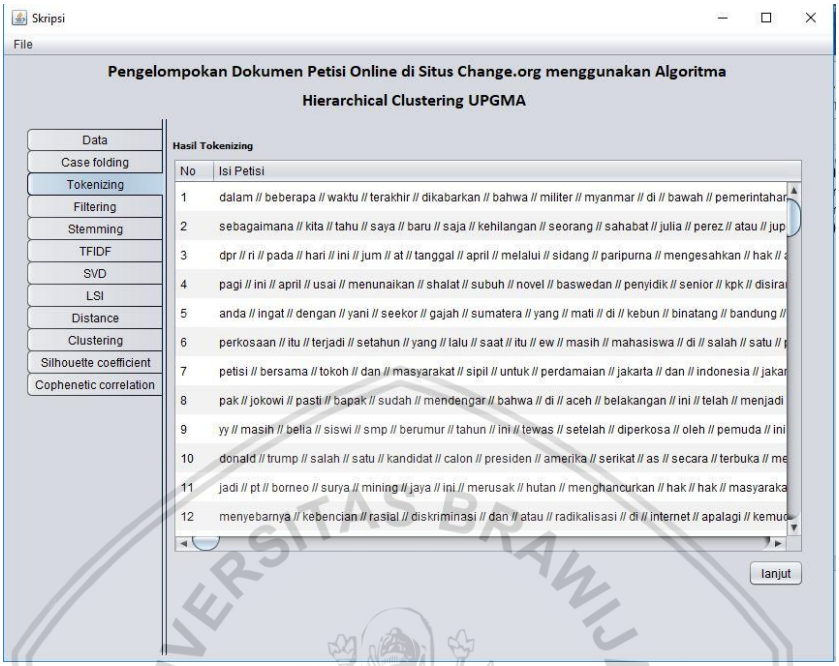
Gambar 5.3 Implementasi Antarmuka Lowecase



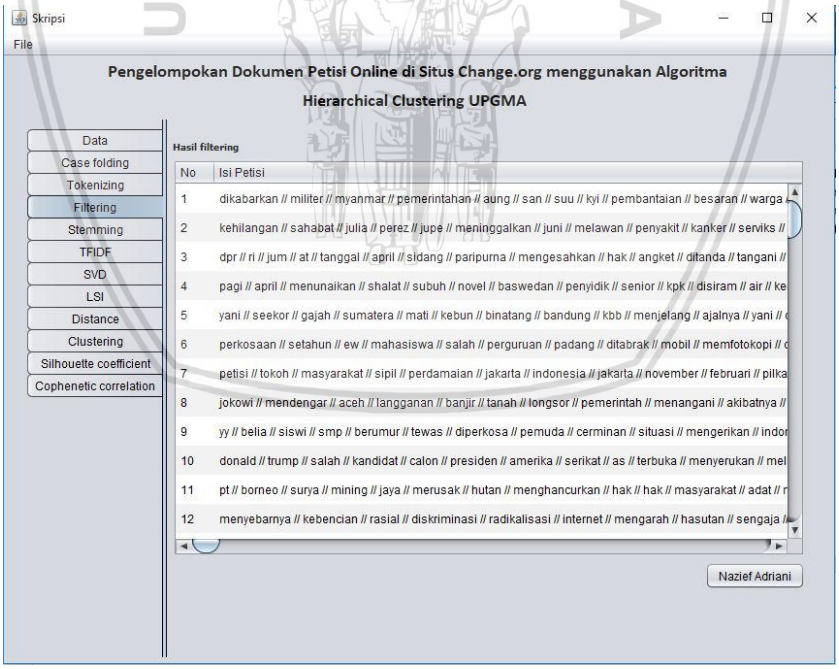
Gambar 5.4 Implementasi Antarmuka Hapus Karakter Non Abjad

Pada halaman antarmuka menu **Tokenizing** dan **Filtering**, hasil proses *Tokenizing* dan *Filtering* akan ditampilkan pada tabel. Pada halaman ini juga terdapat tombol lanjut yang digunakan untuk berpindah halaman dan memproses

data dokumen pada halaman antarmuka selanjutnya. Halaman antarmuka **Tokenizing** ditunjukan pada Gambar 5.5. Sedangkan halaman antarmuka *Filtering* ditunjukan pada Gambar 5.6.

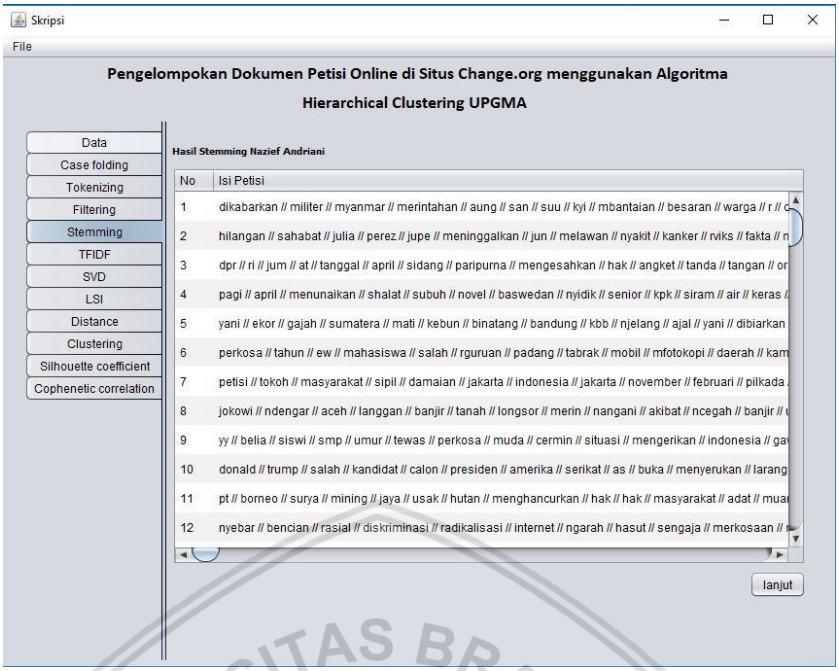


Gambar 5.5 Implementasi Antarmuka *Tokenizing*



Gambar 5.6 Implementasi Antarmuka *Filtering*

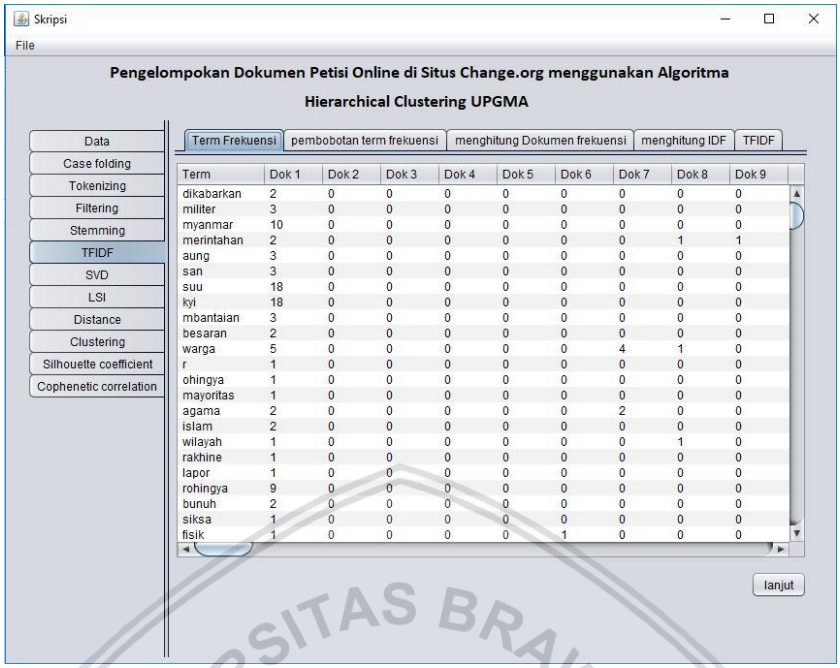
Pada halaman antarmuka menu **Stemming**, data hasil proses pembentukan kata dasar akan ditampilkan pada tabel berjudul hasil *Stemming* nazief andriani. Sistem juga menyimpan isi hasil proses *Stemming* kedalam variabel untuk diproses pada menu selanjutnya. Halaman antarmuka *Stemming* ditunjukan pada Gambar 5.7.



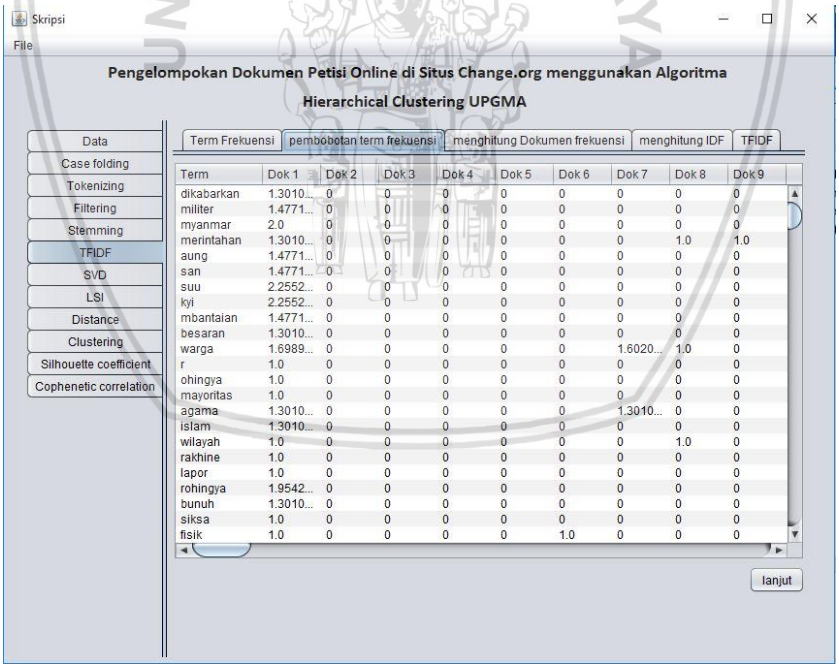
Gambar 5.7 Implementasi Antarmuka *Stemming*

Setelah proses *Text Preprocessing* telah dilakukan, selanjutnya adalah lanjut pada halaman antarmuka pembobotan TFIDF. Pada halaman ini dokumen petisi yang di*input* akan diubah menjadi numerik. Hasil pada dokumen antarmuka *Stemming* akan dihitung banyaknya kata pada setiap dokumen. Hasil tersebut akan ditampilkan pada tabel di menu **TFIDF** dan di *sub* menu **term frekuensi**. Halaman antarmuka *term* frekuensi ditunjukan pada Gambar 5.8. Pengguna dapat melanjutkan dengan menekan tombol lanjut pada bagian kanan bawah antarmuka halaman.

Setelah menekan tombol lanjut, maka sistem akan menampilkan bagian antarmuka halaman **pembobotan term frekuensi**. Hasil perhitungan *term* frekuensi dihalaman sebelumnya akan di hitung berdasarkan rumus Persamaan 2.1 dan hasilnya akan ditampilkan pada tabel di *sub* menu **pembobotan term frekuensi**. Halaman antarmuka **pembobotan term frekuensi** ditunjukan pada Gambar 5.9. Pengguna dapat lanjut ke proses selanjutnya dengan menekan tombol lanjut.



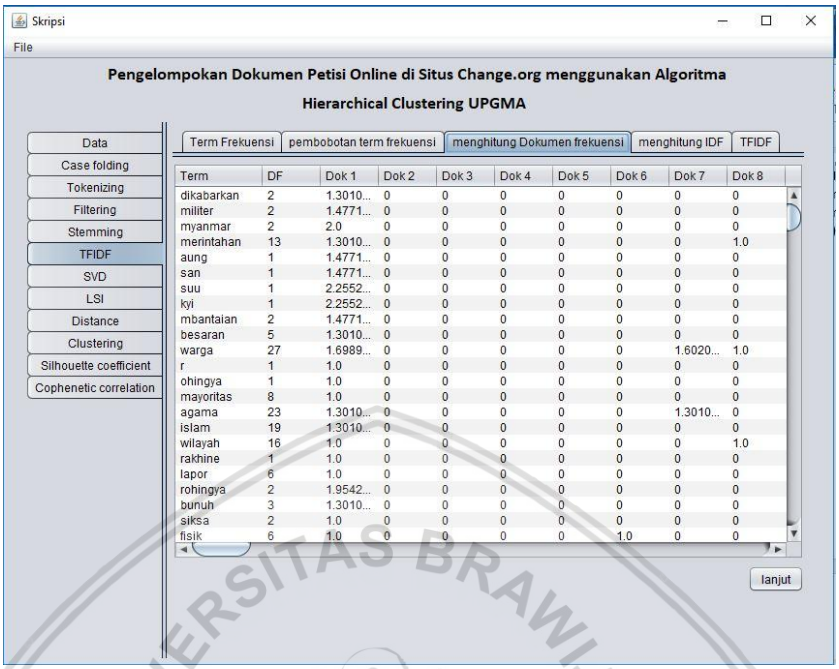
Gambar 5.8 Implementasi Antarmuka *Term* Frekuensi



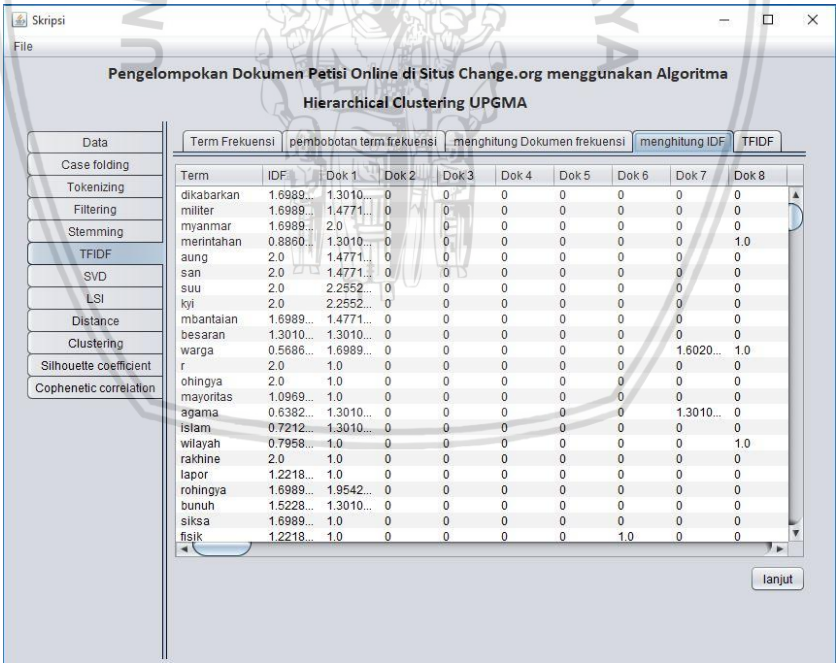
Gambar 5.9 Implementasi Antarmuka Pembobotan *Term* Frekuensi

Pada menu TFIDF masih terdapat 3 *sub* menu lainnya, yaitu **menghitung dokumen frekuensi, menghitung IDF** dan **menghitung TFIDF**. Pada ketiga halaman antarmuka tersebut, berisikan tabel untuk menunjukan hasil perhitungan dan tombol lanjut untuk melanjutkan proses dihalaman antarmuka selanjutnya. Halaman antarmuka *sub* menu **dokumen frekuensi** ditunjukan pada Gambar 5.10,

halaman antarmuka *sub* menu **menghitung IDF** ditunjukan pada Gambar 5.11 dan halaman antarmuka *sub* menu **menghitung TFIDF** ditunjukan pada Gambar 5.12.

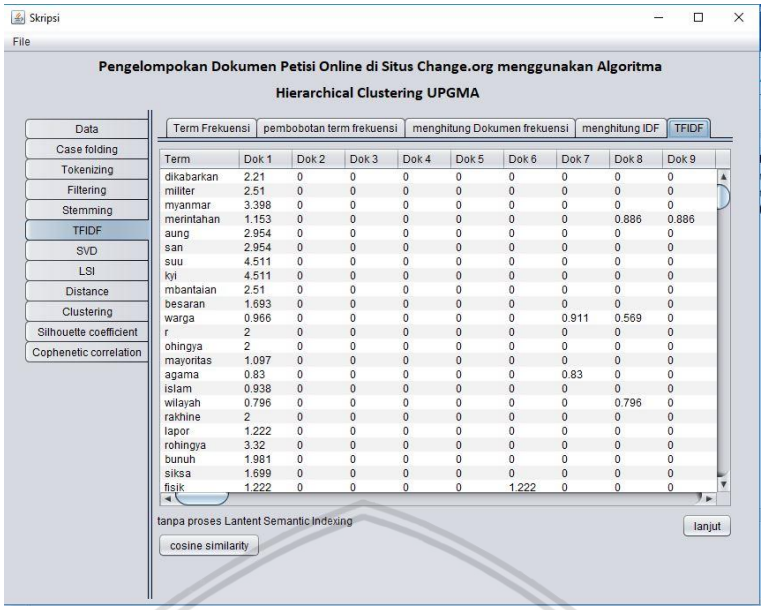


Gambar 5.10 Implementasi Antarmuka Perhitungan DF



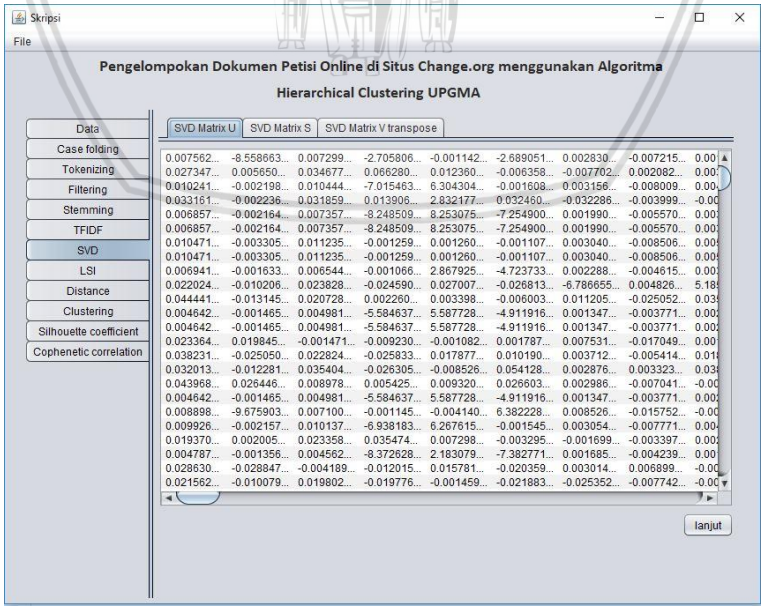
Gambar 5.11 Implementasi Antarmuka Pembobotan IDF

Berbeda ketika pengguna menekan tombol lanjut, Pada *sub* menu pembobotan TFIDF terdapat tombol **lanjut tanpa proses LSI**. Tombol tersebut akan lansung menampilkan menu **Cosine similarity** tanpa melewati menu **SVD** dan menu **LSI**. Pada tabel yang menampilkan hasil TFIDF, *header* tabel yang ditampilkan adalah dokumen sedangkan pada baris pertama menampilkan susunan *term* atau kata.

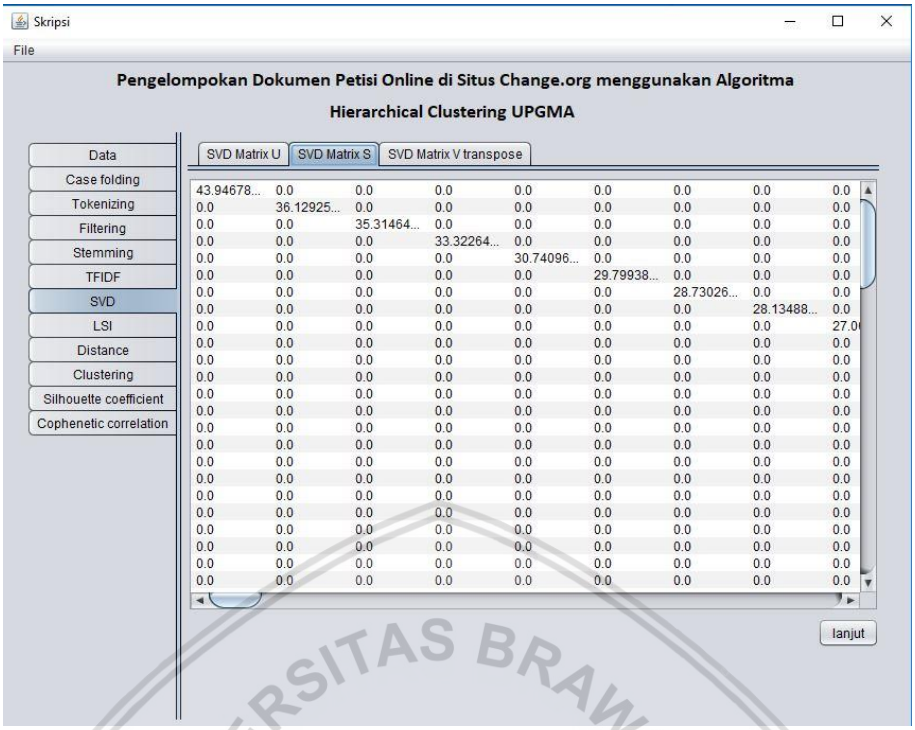


Gambar 5.12 Implementasi Antarmuka Pembobotan TFIDF

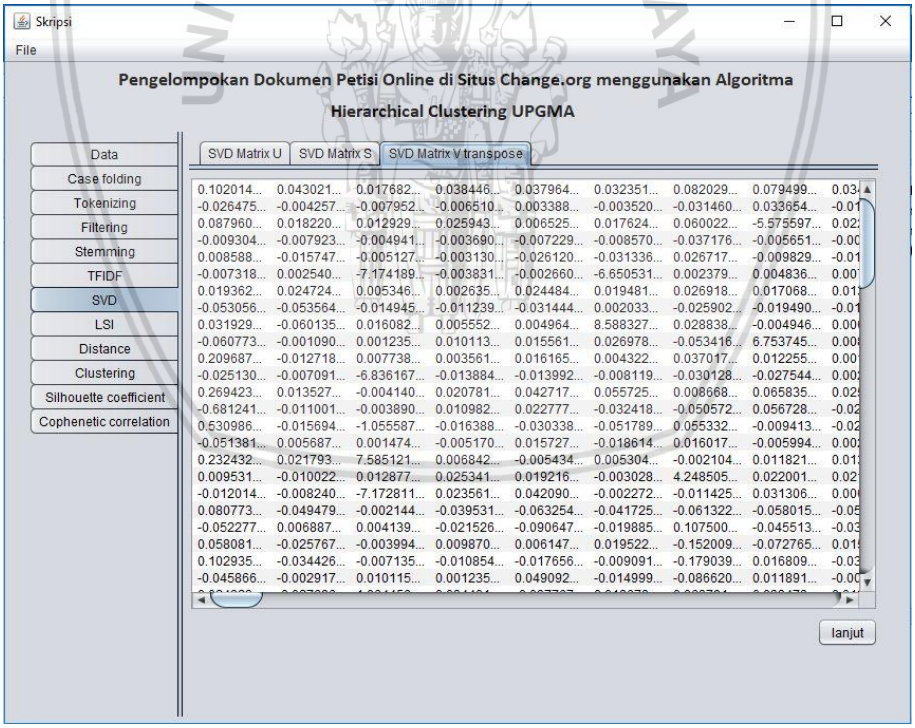
Sub menu SVD memiliki 3 *sub* menu didalamnya, terdiri dari *sub* menu **SVD matrik U**, *sub* menu **SVD matrik S** dan *sub* menu **SVD matrik V *transpose***. Pada ketiga *sub* menu tersebut sistem menampilkan tabel yang tidak memiliki *header*. Tabel tersebut menunjukan implementasi matrik hasil pemotongan matrik TFIDF. Ketiga *sub* menu juga terdapat tombol lanjut yang dapat digunakan pengguna untuk beralih ke halaman antarmuka selanjutnya. Halaman antarmuka *sub* menu **SVD matrik U** ditunjukan pada gambar 5.13. Halaman antarmuka *sub* menu **SVD matrik S** ditunjukan pada Gambar 5.14. Halaman antarmuka *sub* menu **SVD matrik V transpose** ditunjukan pada Gambar 5.15.



Gambar 5.13 Implementasi Antarmuka Pembentukan Matrik U



Gambar 5.14 Implementasi Antarmuka Pembentukan Matrik S



Gambar 5.15 Implementasi Antarmuka Pembentukan Matrik V^T

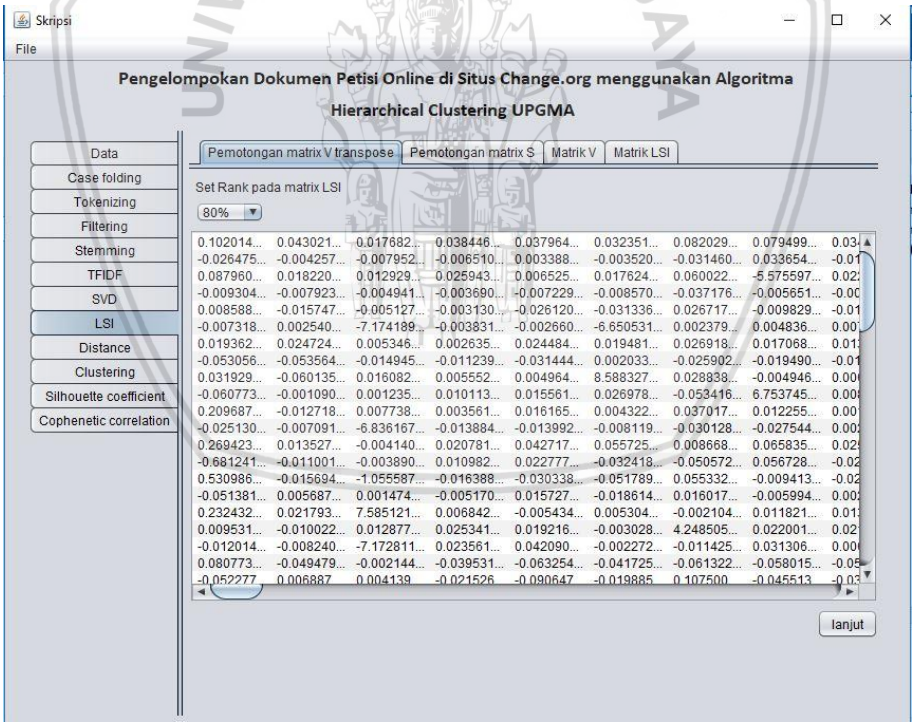
Setelah pengguna menekan tombol lanjut pada *sub* menu **SVD matrik V *transpose***, maka sistem akan mengalihkan pada menu selanjutnya yaitu menu **LSI**. Pada menu **LSI** terdapat 4 *sub* menu yang terdiri dari, **pemotongan matrik V *transpose***, **pemotongan matrik S**, **matrik V** dan **matrik LSI.**

Sub menu yang pertama, sistem menampilkan sebuah tabel dan terdapat tombol pilihan di atasnya. Tombol tersebut berfungsi untuk menentukan *rank* yang akan digunakan pada pemotongan matrik hasil SVD. Pilihan pada tombol mulai dari *rank* 5% hingga *rank* 100%. Sistem baru dapat menampilkan hasil pemotongan ketika pengguna telah menekan pada tombol pilihan *rank* tersebut. Pada *sub* menu **pemotongan matrik V transpose** tabel tidak menampilkan *header* yang menunjukan nilai angka pada tabel tersebut adalah berupa matrik. Pada bagian bawah kanan terdapat tombol lanjut untuk memproses dan mengalihkan pada halaman antarmuka selanjutnya. Halaman antarmuka **menu pemotongan matrik V transpose** ditunjukan pada Gambar 5.16.

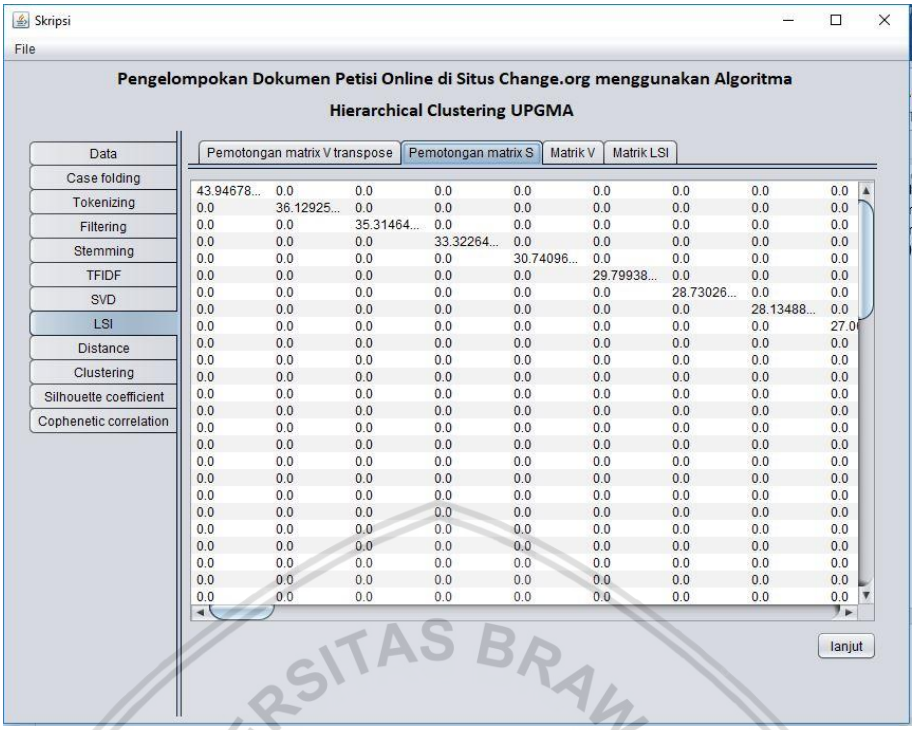
Sub menu kedua terdapat *sub* menu **matrik S**. Halaman antarmuka ini menampilkan tabel hasil pemotongan pada matrik S. Halaman antarmuka pemotongan *sub* menu **matrik S** ditunjukan pada Gambar 5.17.

Sub menu ketiga menampilkan matrik V pada tabel antarmuka *sub* menu matrik V. Halaman antarmuka *sub* menu **matrik V** ditunjukan pada Gambar 5.18.

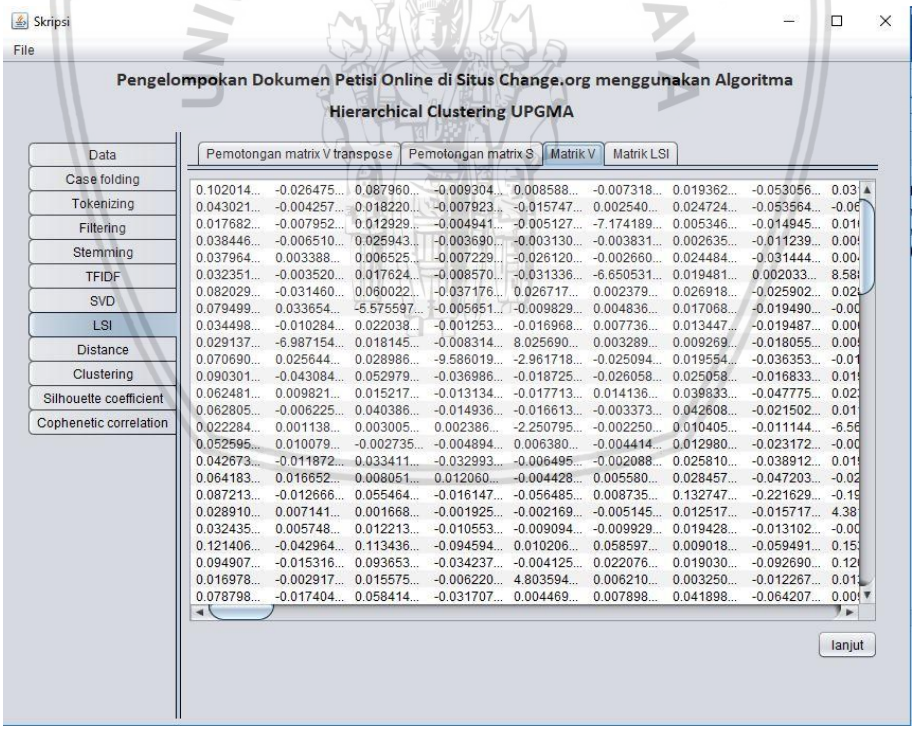
Sub menu keempat terdapat *sub* menu **matrik LSI**. Halaman antarmuka ini menampilkan tabel hasil perkalian matrik V dan matrik S pada *sub* menu halaman sebelumnya. Matrik LSI menggunakan rumus Persamaan 2.5 dalam proses perhitungannya. Halaman antarmuka **matrik LSI** ditunjukan pada Gambar 5.19.



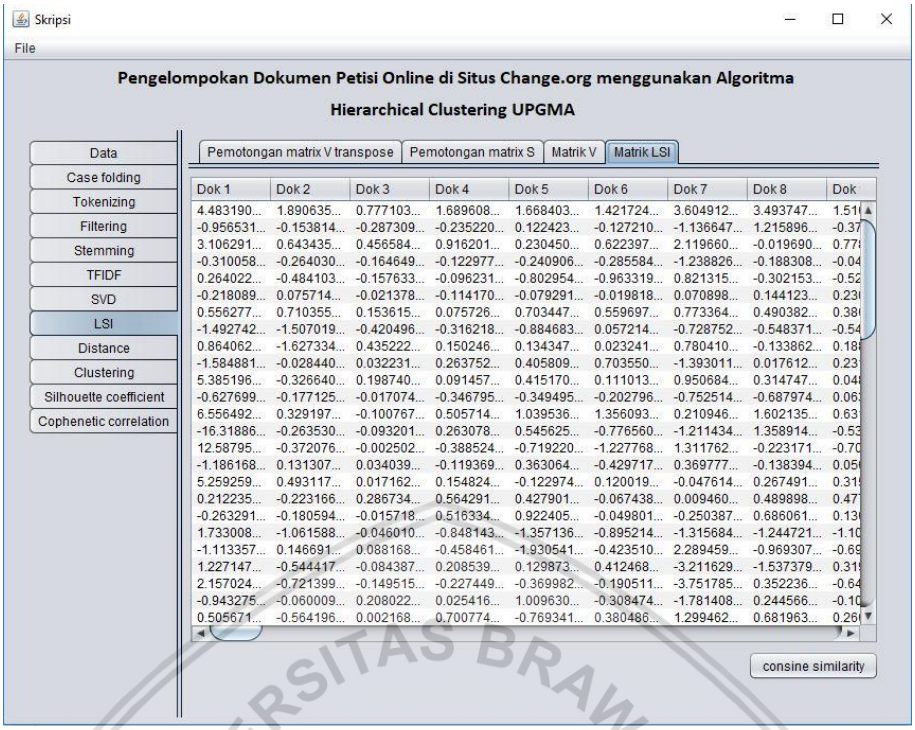
Gambar 5.16 Implementasi Antarmuka Reduksi Matrik V^T



Gambar 5.17 Implementasi Antarmuka Reduksi Matrik S

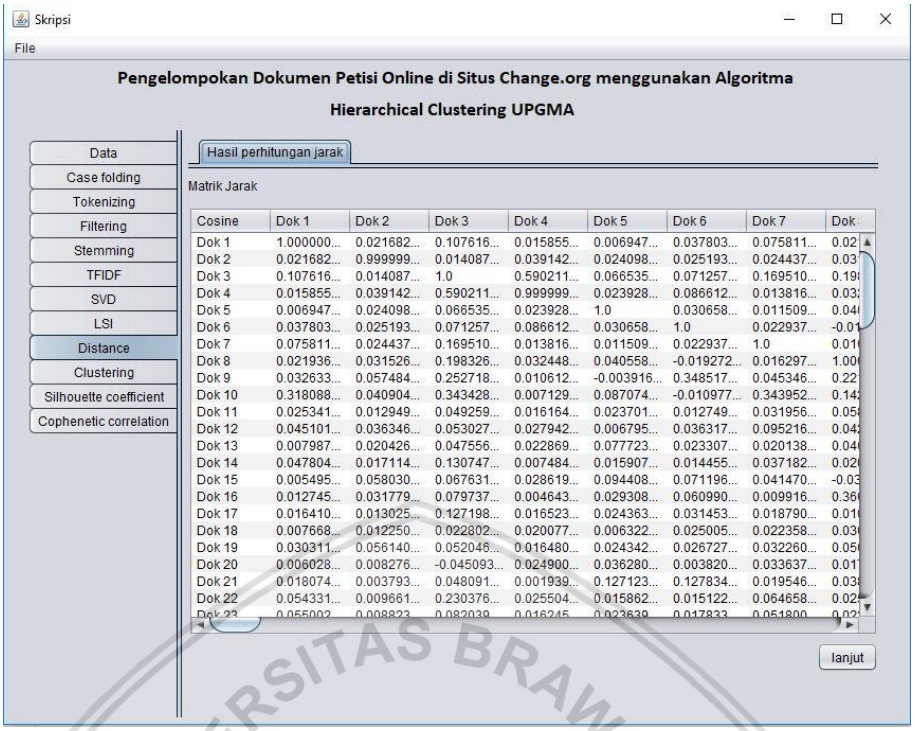


Gambar 5.18 Implementasi Antarmuka Reduksi Matrik V



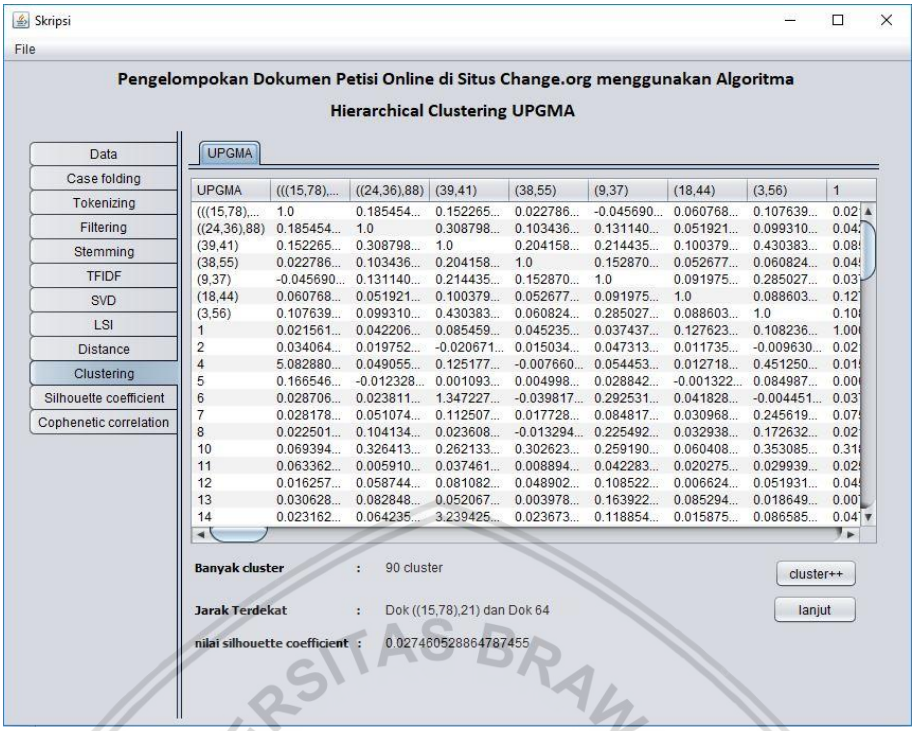
Gambar 5.19 Implementasi Antarmuka Matrik LSI

Halaman antarmuka sistem selanjutnya adalah pada menu **cosine similarity**. Pada halaman antarmuka ini terdapat satu tabel yang menampilkan hasil perhitungan jarak pada semua dokumen *input* pengguna. Tabel yang ditampilkan memiliki *header* yang menunjukan banyaknya dokumen. Jumlah baris dan kolomnya mengikuti jumlah dokumen yang di *input* sebelumnya. Terdapat tombol lanjut pada bagian kanan bawah halaman untuk lanjut pada menu selanjutnya yaitu menu **clustering**. Halaman antarmuka menu **cosine similarity** ditunjukan pada Gambar 5.20.

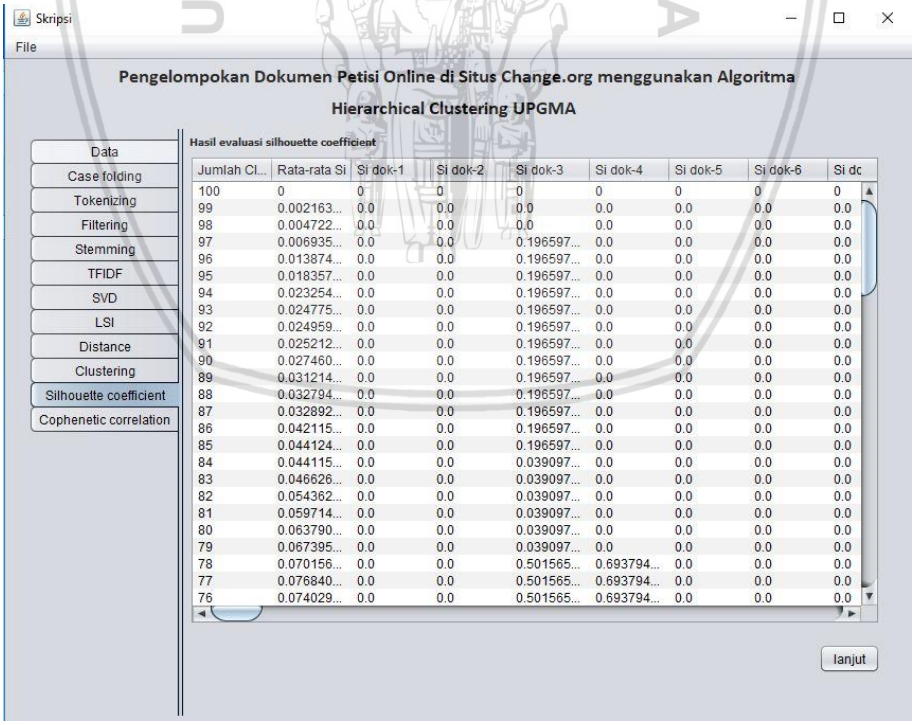


Gambar 5.20 Implementasi Antarmuka Cosine Similarity

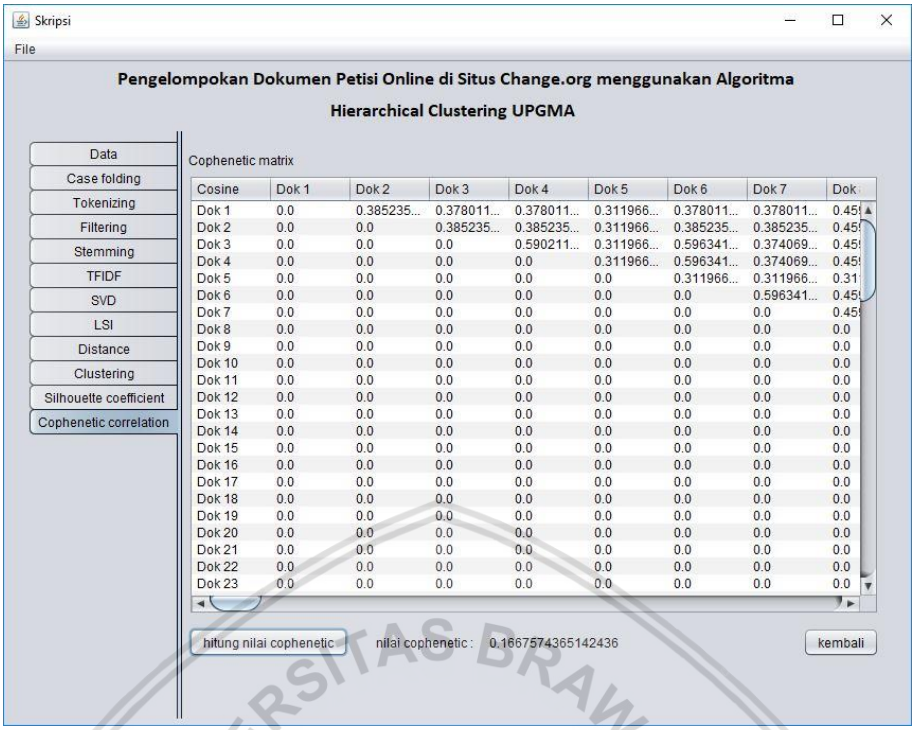
Halaman selanjutnya adalah halaman antarmuka menu **clustering**. Halaman ini merupakan proses utama dari proses dihalaman-halaman sebelumnya. Menu *cluster*ing menampilkan tabel hasil *clustering* dengan rumus perhitungan pada Persamaan 2.7. Bagian kiri bawah halaman antarmuka terdapat 3 label yang terdiri dari label **banyak cluster**, label **jarak terdekat** dan label **nilai Silhouette Coefficient**. Label tersebut menampilkan nilai yang sesuai dengan data hasil *cluster* yang ditampilkan pada tabel. Pada bagian kanan halaman antarmuka terdapat 2 tombol yang terdiri dari **cluster**++ dan tombol **lanjut**. Pada tombol ***cluster***++, ketika pengguna menekan tombol ini maka sistem akan memproses dan mengupdate tabel untuk menghitung proses *cluster*ing pada iterasi selanjutnya, dan akan mengurangi jumlah baris dan kolom tabel sebanyak 1. Sedangkan untuk tombol lanjut, sistem akan mengalihkan halaman antarmuka pada menu selanjutnya. Halaman antarmuka menu **clustering** ditunjukan pada Gambar 5.21. Sedangkan untuk antarmuka menu **evaluasi** ditunjukan pada Gambar 5.22 dan Gambar 5.23.



Gambar 5.21 Implementasi Antarmuka *Cluster*ing UPGMA



Gambar 5.22 Implementasi Antarmuka Evaluasi *Silhouette Coefficient*



Gambar 5.23 Implementasi Antarmuka Evaluasi *Cophenetic*

BAB 6 PENGUJIAN DAN ANALISIS

Pada bab pengujian dan analisis ini berisikan tentang pembahasan mengenai pengujian yang dilakukan oleh sistem, kemudian akan dilakukan analisis sebagai penjelasan dari hasil pengujian tersebut. Terdapat 2 pengujian yang dilakukan, yaitu menentukan nilai *Cophenetic Correlation Coefficient* terbaik dari proses reduksi ataupun tanpa proses reduksi matrik. Pengujian selanjutnya dengan menentukan nilai *Silhouette* pada seluruh level *cluster* dari hasil analisis *rank* yang memiliki nilai *cophenetic* tertinggi.

6.1 Pengujian *Cophenetic Correlation Coefficient*

Pengujian *Cophenetic Correlation Coefficient* pada *cluster* dilakukan untuk melihat kualitas (*quality*) hasil analisis pada *clustering* berhirarki. *Cophenetic Correlation Coefficient* merupakan koefisien korelasi antara matrik jarak dan matrik jarak *dendrogram* (matriks *cophenetic*).

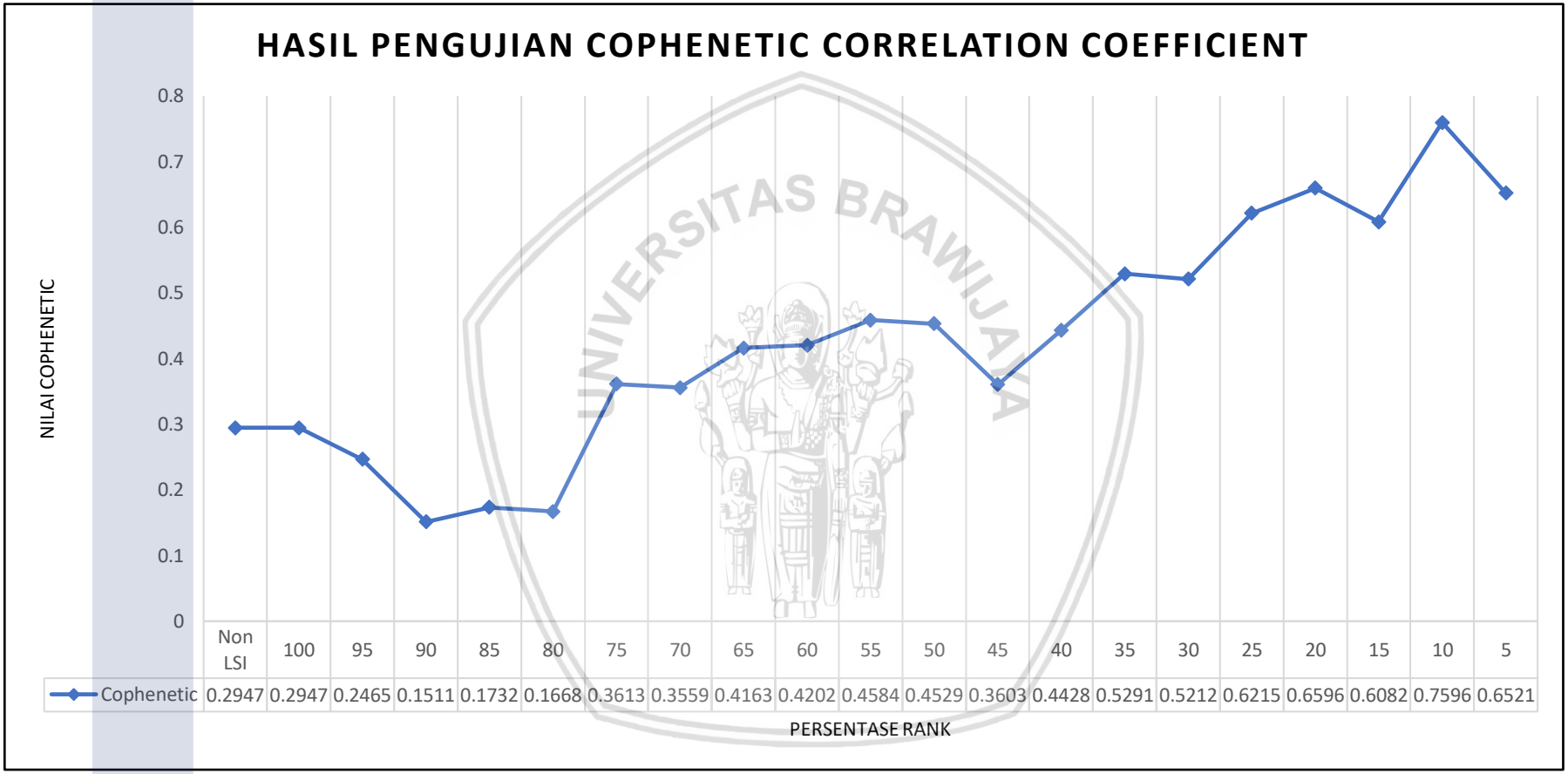
6.1.1 Skenario pengujian *Cophenetic Correlation Coefficient*

Pada pengujian *Cophenetic Correlation Coefficient*, Nilai *cophenetic* pada masing-masing skenario pengujian *rank* dipilih dengan nilai *cophenetic* tertinggi. Matrik LSI yang digunakan mulai dari *rank* 100% hingga direduksi sampai 5%. Pengujian ini juga menentukan nilai *cophenetic* tanpa proses reduksi matrik LSI (*Non* LSI). Hasil dari pengujian *Cophenetic Correlation Coefficient* terdapat pada Tabel 6.1 dan grafik pengujian *Cophenetic Correlation Coefficient* yang ditampilkan dalam Gambar 6.1.

Tabel 6.1 Pengujian Nilai *Cophenetic Correlation Coefficient*

No	*Rank*	Nilai *Cophenetic*
1	Non LSI	0,294716931
2	100%	0,294716931
3	95%	0,246506428
4	90%	0,151136038
5	85%	0,173188879
6	80%	0,16681723
7	75%	0,3613307
8	70%	0,355914879
9	65%	0,416264011
10	60%	0,420214809
11	55%	0,458390911
12	50%	0,452918032
13	45%	0,360295577
14	40%	0,442793319
15	35%	0,529064912

16	30%	0,521216378
17	25%	0,621496405
18	20%	0,659561778
19	15%	0,608184353
20	**10%**	**0,759598426**
21	5%	0,652120738



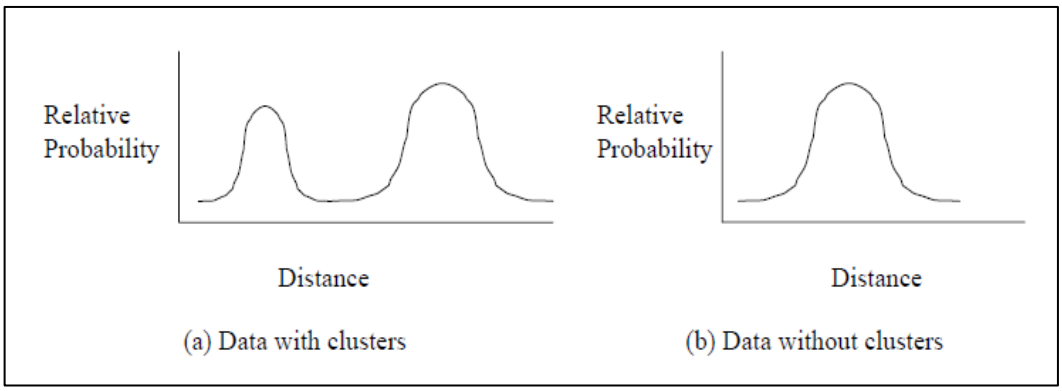
Gambar 6.1 Grafik Pengujian *Cophenetic Correlation Coefficient*

6.1.2 Analisis pengujian *Cophenetic Correlation Coefficient*

Berdasarkan pada Gambar 6.1 dari hasil pengujian *Cophenetic Correlation Coefficient*, maka dapat dilakukan analisis pada pengujian *Cophenetic Correlation Coefficient* nilai *rank*=10% adalah nilai optimal untuk melakukan reduksi matrik LSI pada proses *clustering*. Pada penggunaan nilai *rank* matrik LSI 10%, didapatkan nilai *cophenetic* sebesar 0,759598426. Sedangkan untuk nilai *rank* yang memiliki nilai *cophenetic* terkecil adalah *rank*=90% dengan nilai *cophenetic* sebesar 0,151136038.

Dari hasil analisis tersebut didapatkan bahwa nilai *cophenetic* tanpa menggunakan *Latent Semantic Indexing* memiliki nilai yang kecil, dengan nilai *cophenetic* sebesar 0,294716931. Hal tersebut dikarenakan terdapat permasalahan pembentukan *cluster* pada data yang berdimensi tinggi. Setiap petisi *online* memiliki jumlah kata yang cukup banyak, contohkanlah pada data petisi pertama yang berjudul "Cabut Nobel Perdamaian Aung San Suu Kyi", Petisi tersebut memiliki jumlah sebanyak 587 kata sebelum dilakukan proses *teks preprocessing*. Sehingga pada matriks TFIDF terdapat banyak *term* yang bernilai 0 pada 100 dokumen yang dijadikan data *input*. *Term* yang bernilai 0 ini biasa disebut dengan *noise.*

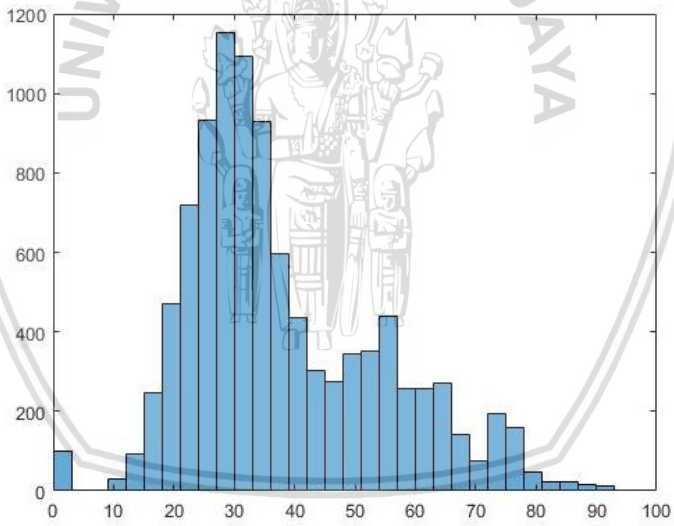
Menurut penelitian yang dilakukan Steinbach, Ertoz, dan Kumar (2004). Untuk tujuan pengelompokan, aspek yang paling relevan pada data berdimensi tinggi adalah menyangkut efek peningkatan dimensi pada matrik jarak atau dalam hal ini matrik *cosine similarity.* Secara khusus, kebanyakan teknik *clustering* bergantung pada matrik jarak dan mengharuskan *object-object* dalam satu *cluster* memiliki jarak yang lebih dekat satu sama lain daripada *object* pada *cluster* lainnya. Adapun cara yang ditawarkan pada penelitian tersebut untuk mengetahui apakah data berdimensi tinggi akan sulit dikelompokkan atau sebaliknya, dengan cara menghitung nilai probabilitas kepadatan data dalam bentuk *histogram*. Jika data memiliki kelompok atau *cluster*, maka *histogram* akan menampilkan 2 gundukan atau puncak. Kedua puncak tersebut mewakili jarak antara *object* dalam *cluster*, dan puncak yang mewakili jarak rata-rata antara *object-object*. Sebaliknya, jika data tidak memiliki *cluster* akan menampilkan satu puncak atau 2 puncak yang berdekatan, maka data akan sulit untuk dilakukan pengelompokan dengan matrik jarak. Contoh *histogram* yang menampilkan data dengan *cluster* dan data tanpa *cluster* ditunjukan pada Gambar 6.2.



Gambar 6.2 *Histogram* data dengan *cluster* dan tanpa *cluster*

Sumber: Steinbach, Ertoz, dan Kumar (2004)

Berdasarkan teknik tersebut penulis menggunakan alat bantu aplikasi *Matlab* untuk menghitung nilai *probabilitas* kepadatan data dan menampilkannya dalam bentuk *histogram*. Data untuk menghitung *histogram* tersebut menggunakan matrik *term frekuensi* tanpa proses LSI, kemudian dilanjutkan dengan menghitung matrik jarak *cosine similarity*, hasil perhitungan probabilitas kepadatan data pada 100 data petisi *online* disitus change.org dapat dilihat pada Gambar 6.3.



Gambar 6.3 *Histogram* probabilitas kepadatan data 100 petisi *online*

Pada Gambar 6.3 tersebut dapat dianalisis bahwa 100 data petisi *online* yang digunakan hanya memiliki satu gundukan (*peak*) dominan, mengartikan bahwa data petisi *online* sebanyak 100 dokumen tanpa reduksi matrik yang digunakan memiliki dimensi yang tinggi dan memiliki tingkat kesulitan dalam melakukan *clustering* dokumen. Hal tersebut ditunjukan dengan nilai *cophenetic* yang bernilai kecil sebesar 0,294716931.

Berdasarkan hasil analisis pengujian *Cophenetic Correlation Coefficient* didapatkan bahwa penggunaan reduksi matrik menggunakan *Latent Semantic Indexing* dapat membantu dan meningkatkan performansi pada data berdimensi tinggi. Pada grafik Gambar 6.1 menunjukkan nilai *cophenetic* sebesar 0,759598426

pada skenario penggunaan *rank* matrik LSI sebanyak 10%. Pada *Latent Semantic Indexing* menjelaskan bahwa reduksi matrik akan semakin tinggi performansinya pada persentase tertentu, akan tetapi jika terlalu banyak reduksi juga tidak menunjukan performansi yang baik. Hal tersebut ditunjukan pada penggunaan *rank* matrik LSI sebanyak 5%, pada skenario tersebut nilai *cophenetic* justru menurun pada nilai 0,652120738.

Pada reduksi matrik LSI, hasil reduksi merepresentasikan nilai kesesuaian dokumen dengan *term*. Matrik S menandakan nilai *singular*, yang mana nilai *singular* diurutkan secara *descending* yang mengartikan semakin besar *rank* matrik maka nilai *singular*nya kecil. Begitupun dengan matrik LSI, semakin besar *rank*nya semakin banyak *noise* yang digunakan. Tetapi jika dilakukan pemotongan matrik terlalu banyak, data yang merepresentasikan dokumen justru terhapus dan hasil performasi *cluster*ing bernilai rendah seperti pada *rank* 5% yang memiliki nilai *cophenetic* sebesar 0,652120738.

6.2 Pengujian *Silhouette Coefficient*

Pengujian *Silhouette Coefficient* dilakukan untuk mengetahui jumlah *cluster* terbaik dengan nilai *Silhouette* tertinggi menggunakan skenario *rank* hasil pengujian *cophenetic* tertinggi.

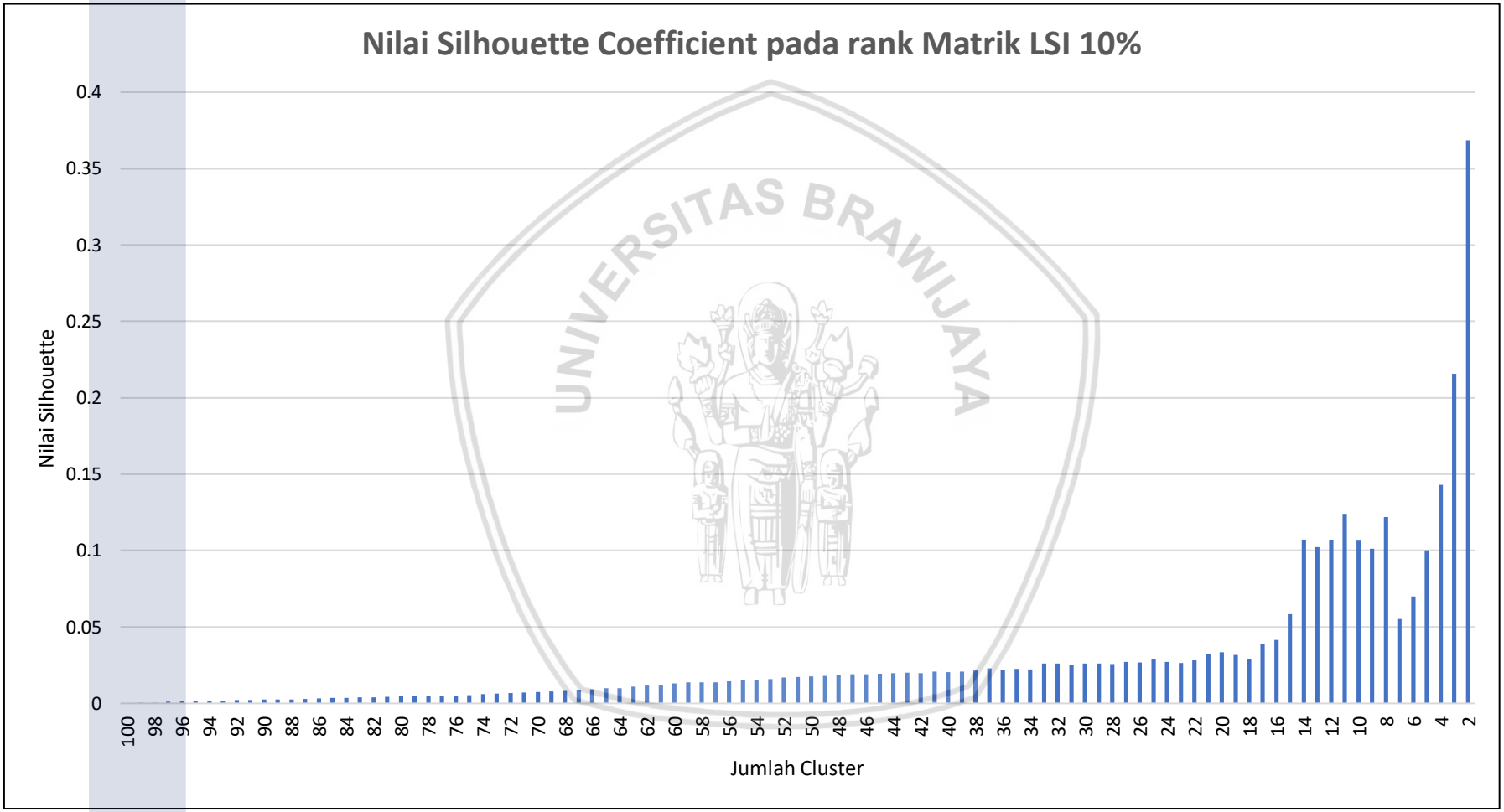
6.2.1 Skenario pengujian *Silhouette Coefficient*

Pada pengujian *Silhouette Coefficient*. Nilai *Silhouette* pada masing masing skenario pengujian jumlah *cluster* dipilih dengan nilai *Silhouette* tertinggi. Jumlah *cluster* yang digunakan mulai dari 100 *cluster* hingga 2 *cluster*. Pada pengujian nilai *Silhouette*, menggunakan *rank* matrik LSI sebanyak 10%. Penggunaan nilai matrik LSI 10% dikarenakan pada pengujian sebelumnya menggunakan *Cophenetic Correlation Coefficient*, nilai *cophenetic* tertinggi bernilai 0,7595 pada skenario *rank* matrik LSI 10%. Hasil dari pengujian *Silhouette Coefficient* pada *rank* matrik LSI 10% terdapat pada Tabel 6.2. dan grafik pengujian *Silhouette Coefficient* yang ditampilkan dalam Gambar 6.2.

Tabel 6.2 Pengujian Nilai *Silhouette* pada *rank* matrik LSI 10%

No	Jumlah *Cluster*	Nilai *Silhouette*	No	Jumlah *Cluster*	Nilai *Silhouette*
1	100 *cluster*	0	51	50 *cluster*	0,0175631
2	99 *cluster*	0,00043595	52	49 *cluster*	0,01805781
3	98 *cluster*	0,00057436	53	48 *cluster*	0,01877048
4	97 *cluster*	0,00105772	54	47 *cluster*	0,01891551
5	96 *cluster*	0,00132612	55	46 *cluster*	0,01912187
6	95 *cluster*	0,00144298	56	45 *cluster*	0,01946761
7	94 *cluster*	0,00181359	57	44 *cluster*	0,01969162
8	93 *cluster*	0,00192798	58	43 *cluster*	0,0199448
9	92 *cluster*	0,00203962	59	42 *cluster*	0,01961976
10	91 *cluster*	0,002186	60	41 *cluster*	0,0208539

11	90 *cluster*	0,00241841	61	40 *cluster*	0,02062856
12	89 *cluster*	0,00253756	62	39 *cluster*	0,02084908
13	88 *cluster*	0,00262668	63	38 *cluster*	0,02154384
14	87 *cluster*	0,00288163	64	37 *cluster*	0,02280813
15	86 *cluster*	0,00331349	65	36 *cluster*	0,02198404
16	85 *cluster*	0,00355323	66	35 *cluster*	0,02245342
17	84 *cluster*	0,00368727	67	34 *cluster*	0,02218139
18	83 *cluster*	0,00387683	68	33 *cluster*	0,02615728
19	82 *cluster*	0,00410803	69	32 *cluster*	0,02617226
20	81 *cluster*	0,00439163	70	31 *cluster*	0,02505258
21	80 *cluster*	0,00449532	71	30 *cluster*	0,02600486
22	79 *cluster*	0,00453056	72	29 *cluster*	0,02593781
23	78 *cluster*	0,00476793	73	28 *cluster*	0,02575333
24	77 *cluster*	0,00491521	74	27 *cluster*	0,02717202
25	76 *cluster*	0,00512185	75	26 *cluster*	0,02670616
26	75 *cluster*	0,00544796	76	25 *cluster*	0,02872806
27	74 *cluster*	0,00599016	77	24 *cluster*	0,02720693
28	73 *cluster*	0,0063811	78	23 *cluster*	0,02629078
29	72 *cluster*	0,006836	79	22 *cluster*	0,02829266
30	71 *cluster*	0,00712862	80	21 *cluster*	0,03248444
31	70 *cluster*	0,0074195	81	20 *cluster*	0,03336288
32	69 *cluster*	0,00784271	82	19 *cluster*	0,03176715
33	68 *cluster*	0,00824461	83	18 *cluster*	0,02905021
34	67 *cluster*	0,00892418	84	17 *cluster*	0,0389672
35	66 *cluster*	0,00914561	85	16 *cluster*	0,04135449
36	65 *cluster*	0,00987696	86	15 *cluster*	0,05830723
37	64 *cluster*	0,01008844	87	14 *cluster*	0,10723146
38	63 *cluster*	0,01103051	88	13 *cluster*	0,10245097
39	62 *cluster*	0,01164427	89	12 *cluster*	0,10678701
40	61 *cluster*	0,01167643	90	11 *cluster*	0,12406919
41	60 *cluster*	0,01325462	91	10 *cluster*	0,1064329
42	59 *cluster*	0,01378922	92	9 *cluster*	0,10109271
43	58 *cluster*	0,01389103	93	8 *cluster*	0,1220149
44	57 *cluster*	0,01389975	94	7 *cluster*	0,05528073
45	56 *cluster*	0,01434084	95	6 *cluster*	0,06980884
46	55 *cluster*	0,01563892	96	5 *cluster*	0,10024922
47	54 *cluster*	0,01515827	97	4 *cluster*	0,14301385
48	53 *cluster*	0,01598562	98	3 *cluster*	0,21574889
49	52 *cluster*	0,01692651	**99**	**2 *cluster***	**0,36466466**
50	51 *cluster*	0,01714265			



Gambar 6.4 Grafik Pengujian *Silhouette Coefficient*

6.2.2 Analisis pengujian *Silhouette Coefficient*

Berdasarkan pada Gambar 6.4 dari hasil pengujian *Silhouette Coefficient*, maka dapat dilakukan analisis Pada pengujian *Silhouette Coefficient* dengan menggunakan *rank* matrik LSI 10% memiliki nilai *Silhouette* tertinggi sebesar 0,36466466. Nilai *silhouette* tertinggi tersebut didapatkan pada skenario jumlah *cluster* sebanyak 2 *cluster*. Sedangkan untuk nilai *silhouette* terkecil adalah pada jumlah *cluster* sebanyak 100 *cluster* dengan nilai *silhouette* sama dengan 0. Nilai pada jumlah *cluster* sebanyak 100 *cluster* memiliki nilai *silhouette* kecil dikarenakan belum dilakukan pembentukan *cluster* pada setiap dokumen petisi *online*. Setiap *cluster* memiliki nilai *silhouette* 0 dikarenakan setiap *cluster* tersebut hanya terdiri dari 1 *object* atau dokumen saja. Pada *cluster* yang memiliki 1 *object*, maka nilai s(i) atau *silhouette*nya bernilai 0 dan hasil rata-rata pada 100 dokumen akan bernilai 0 pula.

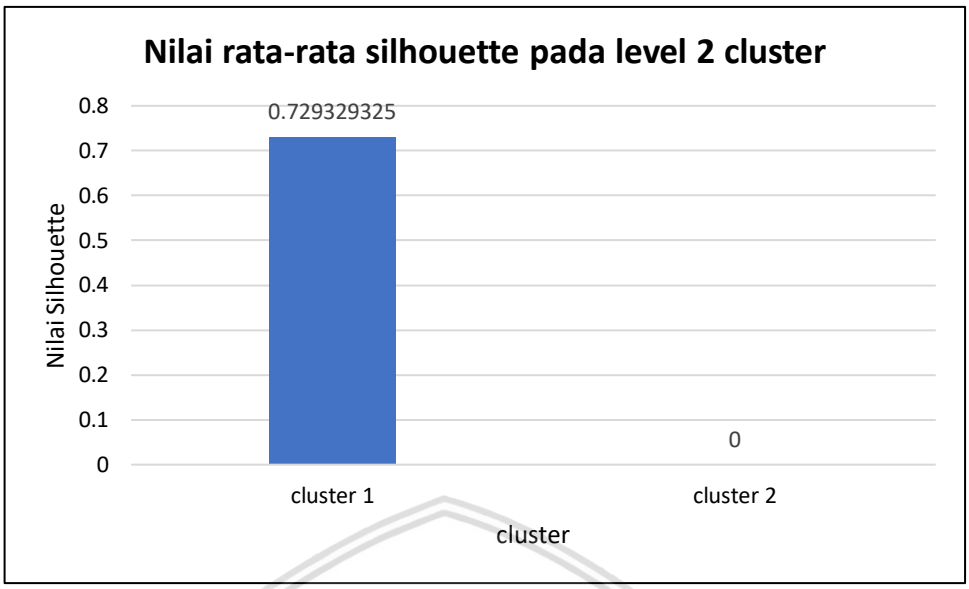
Pada nilai *silhouette* tertinggi, setiap *object* atau dokumen memiliki nilai *silhouette* masing-masing sebelum dirata-rata per *cluster*nya. Tabel 6.3 Merupakan nilai *silhouette* per *object* pada level 2 *cluster* menggunakan *rank* matrik LSI 10%.

Tabel 6.3 Nilai *Silhouette* setiap dokumen *rank* LSI 10% pada level 2 *cluster*

Object	Nilai *Silhouette*	*Cluster*	Object	Nilai *Silhouette*	*Cluster*
Dok 1	0,667945627	*cluster* 1	Dok 51	0,96734222	*cluster* 1
Dok 2	0,771391852	*cluster* 1	Dok 52	0,73544465	*cluster* 1
Dok 3	0,828161721	*cluster* 1	Dok 53	0,45509633	*cluster* 1
Dok 4	0,679141558	*cluster* 1	Dok 54	0,77550552	*cluster* 1
Dok 5	1	*cluster* 1	Dok 55	0,55796078	*cluster* 1
Dok 6	0,824997695	*cluster* 1	Dok 56	0,73528184	*cluster* 1
Dok 7	0,502729419	*cluster* 1	Dok 57	0,81371017	*cluster* 1
Dok 8	0,730023432	*cluster* 1	Dok 58	0,94389578	*cluster* 1
Dok 9	0,693288499	*cluster* 1	Dok 59	0,63905804	*cluster* 1
Dok 10	0,628266218	*cluster* 1	Dok 60	0,4043094	*cluster* 1
Dok 11	0,942845489	*cluster* 1	Dok 61	0,9107623	*cluster* 1
Dok 12	0,787546268	*cluster* 1	Dok 62	0,87462179	*cluster* 1
Dok 13	0,784035159	*cluster* 1	Dok 63	0,82794627	*cluster* 1
Dok 14	0,775513521	*cluster* 1	Dok 64	0,61049184	*cluster* 1
Dok 15	0,879037383	*cluster* 1	Dok 65	0,65695985	*cluster* 1
Dok 16	0,796412411	*cluster* 1	Dok 66	0,87905832	*cluster* 1
Dok 17	0,753452761	*cluster* 1	Dok 67	0,7985388	*cluster* 1
Dok 18	0,797354518	*cluster* 1	Dok 68	0,74871004	*cluster* 1
Dok 19	0,962125246	*cluster* 1	Dok 69	0,85970457	*cluster* 1
Dok 20	0,949208645	*cluster* 1	Dok 70	0,68583932	*cluster* 1
Dok 21	0,947100053	*cluster* 1	Dok 71	0,70267005	*cluster* 1
Dok 22	0,360111462	*cluster* 1	Dok 72	0,67302961	*cluster* 1

Dok 23	0,654543945	*cluster* 1	Dok 73	0,76112336	*cluster* 1
Dok 24	0,499955125	*cluster* 1	Dok 74	0,64165944	*cluster* 1
Dok 25	0,641718359	*cluster* 1	Dok 75	0,92427363	*cluster* 1
Dok 26	0,983953944	*cluster* 1	Dok 76	0,54220545	*cluster* 1
Dok 27	0,706858641	*cluster* 1	Dok 77	0,80691468	*cluster* 1
Dok 28	0,796457856	*cluster* 1	Dok 78	0,9311892	*cluster* 1
Dok 29	0,632116564	*cluster* 1	Dok 79	0,60961834	*cluster* 1
Dok 30	0,71186611	*cluster* 1	Dok 80	0,80336853	*cluster* 1
Dok 31	0,553064967	*cluster* 1	Dok 81	0,65465023	*cluster* 1
Dok 32	0,455770208	*cluster* 1	Dok 82	0,81955199	*cluster* 1
Dok 33	0,835750666	*cluster* 1	Dok 83	0,80858616	*cluster* 1
Dok 34	0,928146593	*cluster* 1	Dok 84	0,79863185	*cluster* 1
Dok 35	0,810905236	*cluster* 1	Dok 85	0,8232173	*cluster* 1
Dok 36	0,40996803	*cluster* 1	Dok 86	0,72128058	*cluster* 1
Dok 37	0,836208048	*cluster* 1	Dok 87	0,7638983	*cluster* 1
Dok 38	0,541296555	*cluster* 1	Dok 88	0,84608444	*cluster* 1
Dok 39	0,63385768	*cluster* 1	Dok 89	0,818242	*cluster* 1
Dok 40	0,836201026	*cluster* 1	Dok 90	0,86381097	*cluster* 1
Dok 41	0,607180807	*cluster* 1	Dok 91	0,82572816	*cluster* 1
Dok 42	0,228582475	*cluster* 1	Dok 92	0,90243068	*cluster* 1
Dok 43	0,386832093	*cluster* 1	Dok 93	0,49026501	*cluster* 1
Dok 44	0,849105605	*cluster* 1	Dok 94	0,49113216	*cluster* 1
Dok 45	0,847455179	*cluster* 1	Dok 95	0,89682241	*cluster* 1
Dok 46	0,622702909	*cluster* 1	Dok 96	0,78756274	*cluster* 1
Dok 47	0,815802617	*cluster* 1	**Dok 97**	**0**	***cluster* 2**
Dok 48	0,62821845	*cluster* 1	Dok 98	0,89495612	*cluster* 1
Dok 49	0,50285185	*cluster* 1	Dok 99	0,7325907	*cluster* 1
Dok 50	0,823802723	*cluster* 1	Dok 100	0,9046608	*cluster* 1

Berdasarkan Tabel 6.3, dapat dianalisis bahwa *cluster* 1 memiliki nilai *silhouette* yang cukup tinggi pada setiap *object* atau dokumennya. Hal tersebut berbanding terbalik dengan nilai *silhouette* yang terdapat pada *cluster* 2 yang hanya memiliki *object* pada dokumen 97 saja. Seperti halnya pada pembentukan *cluster* sebanyak 100 dokumen, pembentukan *cluster* yang memiliki *object* sebanyak satu (*singleton cluster*) akan bernilai 0 pada *silhouette*nya, sehingga ketika dirata-rata dengan nilai *silhouette* pada *cluster* 1 akan memiliki nilai *silhouette* yang kecil. Gambar 6.5 Merupakan perbandingan rata-rata nilai *silhouette* pada *cluster* 1 dan *cluster* 2.



Gambar 6.5 Rata-rata nilai *Silhouette* pada level 2 *cluster*

Berdasarkan Gambar 6.5 dapat dianalisis bahwa nilai rata-rata *silhouette* pada *cluster* 1 bernilai 0,7293 yang merupakan *cluster* yang cukup kuat. Berbeda dengan *cluster* 2 yang memiliki nilai *silhouette* sebesar 0 masuk pada kategori data tidak memiliki *cluster*. Pada *clustering* dengan menggunakan UPGMA, maka akan dapat dihitung jarak rata-rata pada *cluster* 1 dengan *cluster* 2 (dokumen 97). Adapun jarak *cluster* 1 dengan *cluster* 2 menggunakan rata-rata jarak *cosine similarity* pada UPGMA *clustering* sebesar 0.18564194. Nilai jarak *cluster* 1 dengan *cluster* 2 memiliki nilai yang kecil dan berjarak cukup jauh dengan *cluster* 1, kemudian penulis mencoba membandingkan jarak tersebut dengan jarak rata-rata pada 100 dokumen. Jarak rata-rata seluruh dokumen menggunakan *cosine similarity* berjarak 0,6776, maka dapat disimpulkan bahwa *cluster* 2 atau dokumen 97 memiliki jarak *object* yang jauh dari *cluster* ke 1 dan dapat dikategorikan sebagai *outlier*. *Outlier* merupakan sehimpunan data yang memiliki sifat berbeda dibandingkan dengan kebanyakan data yang ada.

BAB 7 PENUTUP

Pada bab penutup ini berisikan kesimpulan akhir dari penelitian yang telah dilakukan dan juga saran untuk penelitian dan pengembangan selanjutnya.

7.1 Kesimpulan

Dari hasil pengujian dan analisis sebelumnya, maka dapat diambil kesimpulan pada penelitian mengenai pengelompokan dokumen petisi *online* di situs change.org menggunakan algoritme *hierarchical clustering* UPGMA, yaitu:

1. Metode *hierarchical clustering* UPGMA dapat diterapkan pada pengelompokan dokumen petisi *online* di situs change.org. Data yang dilakukan proses pengelompokan adalah petisi *online* berbahasa Indonesia untuk mengetahui berapa *cluster* atau kelompok yang dapat dibentuk dari 100 data petisi. Pada penelitian ini data petisi *online* yang digunakan adalah berasal dari petisi yang tergolong memiliki basis pendukung besar. Sebelum dilakukan proses *clustering* dokumen, 100 dokumen *input* mengalami beberapa proses diantaranya *text preprocessing*, *Term Weighting*, *Singular Value Decomposition*, *Latent Semantic Indexing* dan perhitungan jarak dengan *cosine similarity*.
2. Kualitas *clustering* yang dihasilkan dari proses pengujian menggunakan *Cophenetic Correlation Coefficient* didapatkan bahwa hasil nilai *cophenetic* tertinggi yaitu 0,759598426 dan *rank* pada matrik LSI yang digunakan sebesar 10%. Berdasarkan hasil tersebut dapat diambil kesimpulan bahwa penggunaan reduksi matrik menggunakan *Latent Semantic Indexing* mempengaruhi dan memperbaiki hasil kualitas *cluster* pada data berdimensi tinggi, sebagaimana ditunjukan dengan nilai *cophenetic* yang rendah sebelum dilakukannya proses *Latent Semantic Indexing* dengan nilai *cophenetic* hanya sebesar 0,294716931.
3. Pada proses pengujian menggunakan *Silhouette Coefficient* didapatkan bahwa nilai *Silhouette* tertinggi yaitu sebesar 0,36862758 dan jumlah *cluster* sebanyak 2 *cluster*. Berdasarkan hasil tersebut dapat disimpulkan bahwa nilai *Silhouette* pada hasil *clustering* dokumen masih bernilai rendah dan masuk pada kualitas *cluster* yang lemah, sebagaimana ditunjukan pula dengan nilai *Silhouette* pada jumlah *cluster* yang lainnya yang berada di bawah nilai tersebut. Hal tersebut dikarenakan pada pembentukan *cluster* pada level 2 *cluster* memiliki *singleton cluster* atau *cluster* yang hanya memiliki 1 *object* saja pada dokumen ke 97. Dokumen 97 ini dapat dikategorikan sebagai *outlier* yang memiliki perbedaan jarak cukup jauh pada data *cluster* lainnya.

7.2 Saran

Berdasarkan penelitian yang sudah dilakukan, masih terdapat beberapa kekurangan yang dapat diperbaiki pada penelitian selanjutnya. Adapun saran untuk penelitian selanjutnya yaitu:

4. Pada tahap *Text Preprocessing* sebaiknya dilakukan perbaikan pada kata baku dan pengulangan *Text Preprocessing* sehingga dapat meningkatkan nilai akurasi.
5. Memperhatikan dan memberikan penanganan pada *outlier* untuk data berdimensi tinggi sebelum dilakukan proses *clustering*.
6. Melakukan perbandingan dengan algoritme *hierarchical clustering* lainnya untuk menghasilkan nilai performansi *clustering* terbaik.
7. Perbaikan atau normalisasi nilai negatif yang dihasilkan pada proses *Latent Semantic Indexing*.

DAFTAR PUSTAKA

Agusta, L., 2009. Perbandingan Algoritme *Stemming* Porter dengan Algoritme Nazief & Adriani untuk *Stemming* Dokumen Teks Bahasa Indonesia.Konferensi Nasional Sistem dan Informatika.Universitas Kristen Satya Wacana.

Alfina, T., Santosa, B., dan Ridho, A., 2012 .Analisis Perbandingan Metode *Hierarchical clustering* , K-means dan Gabungan Keduanya dalam *cluster* Data (Studi kasus: Problem Kerja Praktek Jurusan Teknik Industri ITS*)*.Jurnal Teknik ITS Vol.1 September.

Baker, K., 2013. *Singular Value Decomposition Tutorial*.diakses dari https://datajobs.com/data-science-repo/SVD-Tutorial-[Kirk-Baker].pdf. [Diakses 15 Agustus 2017].

Change.org Inc, 2017. Wadah Dunia Untuk Perubahan, diakses dari https://www.change.org/id/ . [Diakses 14 Agustus 2017] .

Geib, J., 2011. *Latent Semantic Sentence Clustering for Multi-Document* Summarization. UCAM-CL-TR-802. University of Cambridge.

Gullo, F. 2015. *From Patterns in Data to Knowledge Discovery: What Data Mining Can Do*. *Physics Procedia 62* , 18-22.

Husni, Dwi, Y. dan Syarief, M., 2015. *Clustering* Dokumen Web (Berita) Bahasa Indonesia Menggunakan Algoritme K-Means.Jurnal Simantec, Vol.4,No. 3 Juli.

Jiawei, H., Kamber, M., and Pei, J. 2012. *Data mining: Concepts and Techniques Third Edition.* Waltham, MA: Morgan Kaufmann.

Langgeni, D. P., Baizal A., Dan Firdaus Y., 2010. *Cluster*ing Artikel Berita Berbahasa Indonesia Menggunakan Unsupervised Feature Selection. Seminar Nasional Informatika 2010. UPN "Veteran" Yogtakarta.

Lestari, N.A., Darma Putra, I.G., and Cahyawan, A. A., 2013. *Personality Types Classification for Indoensian Text in Partners Searching Websites Using Naïve Bayes Methods*.IJSCSI Internasional Jurnal of Computers Science Issues,10(1),1694-0784.

Manning, C. D., Raghavan, P., dan Schütze, H., 2009. *Introduction to Information Retrieval*. Cambridge, England: Cambridge University Press.

Milatina, Syukur, A., dan Supriyanto, C., 2012. Pengaruh *Text Preprocessing* pada *Clustering* Dokumen Teks Berbahasa Indonesia.Jurnal Teknologi Informasi ,Vol 8 Nomor 1.Universitas Dian Nuswantoro.

Muflikhah, L., Baharudin, B., 2009. *High Performance in Minimizing of Term-Document Matrix Representation for Document Clustering*. Conference on Innovative Technologies in Intelligent Systems and Industrial Applications.

Mustikaningsih, W., 2016. Implikasi Petisi *Online* Terhadap Advokasi Kebijakan Publik Tentang RUU Pilkada Langsung 2014-2015.Jurnal Review Politik.

Purwanti, E., 2015. Klasifikasi Dokumen Temu Kembali Informasi dengan K-Nearest Neghbour.Record and library journal.volume 1,nomor 2.

Rahadian, B. A., Kurnianingtyas, D., Mahardika, D. P., Maghfira, T. N., Cholissodin, I., 2017. Analisis Judul Majalah Kawanku Menggunakan *Clustering* K-Means Dengan Konsep Simulasi *Big data* Pada Hadoop Multi Node *Cluster*.Jurnal Teknologi Informasi dan Ilmu Komputer.Vol. 4 ,No.2 Juni.hlm. 75 – 80.

Saracli et al., 2013. *Comparison of hierarchical cluster analysis methods by cophenetic correlation*. Journal of Inequalities and Applications 2013:203.

Steinbach M., Ertoz L., Kumar V., 2004. The Challenges of *Cluster*ing High Dimensional Data. In: Wille L.T. (eds) New Directions in Statistical Physics. Springer, Berlin, Heidelberg

Tala, F. Z., 2003. *A Study of Stemming Effects on Information Retrieval in* Bahasa Indonesia.Institute for Logic, Language and Computation.Universiteit van Amsterdam.

Wahyudi, R., 2012. Change.org*, Media Sosial untuk Perubahan Sosial*. http://tekno.kompas.com/read/2012/10/15/10090221/Change.org..Media. Sosial.untuk.Perubahan.Sosial. [Diakses 14 Agustus 2017].

Wahyuni, I., Auliya, Y. A., Rahmi, A., Mahmudy, W. F., 2016. *Clustering* Nasabah Bank Berdasarkan Tingkat Likuiditas Menggunakan Hybrid Particle Swarm Optimization dengan K-Means. Jurnal Ilmiah Teknologi dan Informasi ASIA (JITIKA) 10(2): 24-33.

Yim, O., and Ramdeen, K. T., 2015. *Hierarchical Cluster Analysis: Comparison of Three Linkage Measures and Application to Psychological Data*. The Quantitative Methods for Psychology, 11 (1), 8-21.

Zhao, Y. and Karypis, G., 2005. *Hierarchical clustering Algorithms for Document Datasets. Data mining* and Knowledge Discovery, 10: 141–168.